WebSphere Message Broker

Message Flows

Version 6 Release 1

WebSphere Message Broker

Message Flows

Version 6 Release 1

Note
Before you use this information and the product that it supports, read the information in the Notices appendix.

This edition applies to version 6, release 1, modification 0, fix pack 8 of IBM WebSphere Message Broker and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this topic collection

This PDF file has been created from the WebSphere Message Broker Version 6.1 (fix pack 8 update, July 2010) information center topics. Always refer to the WebSphere Message Broker online information center to access the most current information. The information center is periodically updated on the document update site and this PDF and others that you can download from that Web site might not contain the most current information.

The topic content included in the PDF does not include the "Related Links" sections provided in the online topics. Links within the topic content itself are included, but are active only if they link to another topic in the same PDF collection. Links to topics outside this topic collection are also shown, but result in a "file not found "error message. Use the online information to navigate freely between topics.

Feedback: do not provide feedback on this PDF. Refer to the online information to ensure that you have access to the most current information, and use the Feedback link that appears at the end of each topic to report any errors or suggestions for improvement. Using the Feedback link provides precise information about the location of your comment.

The content of these topics is created for viewing online; you might find that the formatting and presentation of some figures, tables, examples, and so on are not optimized for the printed page. Text highlighting might also have a different appearance.

Part 1. Developing message flows

Developing message flows

Design, create and maintain message flows by using the workbench.

A message flow is a sequence of processing steps that run in the broker when an input message is received. The topics in this section describe how to create and maintain message flows.

Concept topics:

Task topics:

- "Configuring aggregation flows" on page 665
- "Creating message collections" on page 684
- "Configuring timeout flows" on page 696
- "Configuring flows to handle WebSphere MQ message groups" on page 706

See also a section of topics that contain reference information about message flows.

The workbench provides a set of toolbar icons that invoke wizards that you can use to create any of the resources associated with message flows, for example, message flow projects and ESQL files. Hold the mouse pointer over each icon to see its function.

The workbench lets you open resource files with other editors. Use only the workbench message flow editor to work with message flow files, because this editor correctly validates all changes that you make to these files when you save the message flow.

When you have completed developing your message flow, deploy it to a broker to start its execution.

Tip: You can debug your message flow by using the flow debugger.

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

Message flows overview

A message flow is a sequence of processing steps that run in the broker when an input message is received.

You define a message flow in the workbench by including a number of message flow nodes, each of which represents a set of actions that define a processing step. The way in which you join the message flow nodes together determine which processing steps are carried out, in which order, and under which conditions. The path that you create between one node and another is known as a connection.

A message flow must include an input node that provides the source of the messages that are processed. You can process the message in one or more ways, and optionally deliver it through one or more output nodes. The message is received as a bit stream, and is converted by a parser into a tree structure that is used internally in the message flow. Before the message is delivered to a final destination, it is converted back into a bit stream. For more information about these conversions, see "Parsers" on page 91 and "The message tree" on page 69.

When you want to exchange messages between multiple applications, you might find that the applications do not understand or expect messages in the same format. You might need to provide some processing between the sending and receiving applications that ensures that both can continue to work unchanged, but can exchange messages successfully.

You define the processing that is required when you create and configure a message flow. You can include built-in nodes, nodes that are supplied by a vendor, nodes that you have created yourself (user-defined nodes), or other message flows (known as subflows).

The processing that you set up determines what actions are performed on a message when it is received, the order in which the actions are completed, and the final destinations of the message. All these actions manage the route that a message takes through a message flow.

You can configure additional properties to make your message flow transactional, or multithreaded. You can also add error paths that ensure every message is handled in an appropriate way.

When you want to run a message flow to process messages, you deploy it to a broker, where it is run in an execution group.

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

The following topics describe the concepts that you need to understand to design, create, and configure a message flow and its associated resources:

- Projects
- Nodes
- "WebSphere Adapters nodes" on page 10
- "IBM Information Management System (IMS)" on page 59
- "Configurable services" on page 66
- Version and keywords
- "Message flow connections" on page 67
- "Threading" on page 68
- "Execution and threading models in a message flow" on page 69
- "The message tree" on page 69
- "Parsers" on page 91
- "Properties" on page 131
- "Message flow transactions" on page 136
- "Broker schemas" on page 139
- "Business-level monitoring" on page 141
- Accounting and statistics data
- Aggregation
- "Message collections" on page 163
- "Converting data with message flows" on page 165
- "User exits" on page 167

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

Message flow projects

A message flow project is a specialized container in which you create and maintain all the resources associated with one or more message flows.

You can create a message flow project to contain a single message flow and its resources, or you can group together related message flows and resources in a single message flow project to provide an organizational structure to your message flow resources.

Message flow project resources are created as files, and are displayed in the project in the Broker Development view. These resources define the content of the message

flow, and additional objects that contain detailed configuration information. For example, a project might contain ESQL modules or message mappings, used by one or more nodes in the message flow.

Import one of the following samples from the Samples Gallery to see how the message flow resources of the sample are stored in a Message Flow project. If the sample has a message set, the message set resources are stored in a Message Set project.

- Video Rental
- Comma Separated Value (CSV)

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

Message flow nodes

A message flow node is a processing step in a message flow. It can be a built-in node, a user-defined node, or a subflow node.

A message flow node receives a message, performs a set of actions against the message, and optionally passes the original message, and none or more other messages, to the next node in the message flow.

A message flow node has a fixed number of input and output points known as terminals. You can make connections between the terminals to define the routes that a message can take through a message flow. Message flow nodes are displayed in the node palette associated with the Message Flow editor. The palette is arranged in categories, which group together nodes that provide related processing; for example, transformation.

Input nodes do not have input terminals. The message flow starts when a message is retrieved from an input device; for example, a WebSphere MQ queue. The message flow ends when none or more output messages have been sent by one or more output nodes, and control returns back to the input node. The input node either commits or rolls back the transaction. Input and output nodes can be protocol-specific, to interact with particular systems such as Web Services.

Most nodes are processing nodes, that you can include between your input and output nodes and connect together to define the flow of control. These nodes typically transform a message from one format to another, or route a message along a particular path, or provide more complex options such as aggregation or filtering.

You can configure a node by setting or changing the values for its properties. Some nodes have *mandatory properties*, for which you must set a value. Other properties must have a value, but are assigned a default value which you can leave unchanged. The remaining properties are *optional properties*; no value is required.

When you develop a message flow, the way in which you set the properties of the nodes in that flow influences the way in which the messages are processed by that flow. For example, by setting properties that define input and output WebSphere MQ queue names, you determine where the message flow receives the message from, and where it delivers the message.

You can also configure nodes by using *promoted properties*; promote one or more node properties to become properties of the message flow that contains those

nodes. You can then change these properties at the flow level, rather than having to update one or more individual nodes. You can also promote equivalent properties from more than one node to the same message flow property; for example, you might use this technique to set, at the flow level, the name of the database that all the nodes in the message flow must connect to.

A subset of node properties are *configurable properties*; that is, you can change their values when you deploy the message flow to a broker for execution. You might find this ability useful if you deploy a message flow to more than one broker, and want it to behave in a slightly different way on each broker. For example, when you deploy the message flow to a test broker, you can set a configurable property to force the flow to interact with a test database. When you deploy the same message flow to a production broker, you can set the same property to the value of a production database, without having to update the message flow itself.

The mode that your broker is working in can affect the types of node that you can use; see Restrictions that apply in each operation mode.

You can add nodes of three types into your message flows:

Built-in node
A built-in node is a message flow node that is supplied by WebSphere Message Broker. The built-in nodes provide input and output, manipulation and transformation, decision making, collating requests, and error handling and reporting functions.

For information about all of the built-in nodes supplied by WebSphere Message Broker, see "Built-in nodes" on page 875.

User-defined node
A user-defined node is an extension to the broker that provides a new message flow node in addition to those supplied with the product. It must be written to the user-defined node API provided by WebSphere Message Broker for both C and Java™ languages. The following sample demonstrates how you can write your own nodes in both C and Java languages.

- User-defined Extension

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

Subflow
A subflow is a directed graph that is composed of message flow nodes and connectors and is designed to be embedded in a message flow or in another subflow. A subflow must include at least one Input node or one Output node. A subflow can be executed by a broker only as part of the message flow in which it is embedded, and therefore cannot be independently deployed.

A message is received by an Input node and processed according to the definition of the subflow. That might include being stored through a Warehouse node, or delivered to another message target, for example through an MQOutput node. If required, the message can be passed through an Output node back to the main flow for further processing.

The subflow, when it is embedded in a main flow, is represented by a subflow node, which has a unique icon. The icon is displayed with the

correct number of terminals to represent the Input and Output nodes that you have included in the subflow definition.

The most common use of a subflow is to provide processing that is required in many places within a message flow, or is to be shared between several message flows. For example, you might code some error processing in a subflow, or create a subflow to provide an audit trail (storing the entire message and writing a trace entry).

For more information, see "Using subflows" on page 196.

A node does not always produce an output message for every output terminal: often it produces one output for a single terminal based on the message received or the result of the operation of the node. For example, a Filter node typically sends a message on either the True terminal or the False terminal, but not both.

If you have connected more than one terminal to another node, the processing in the node determines the order in which the message is propagated to the nodes that it is connected to; you cannot change this order. The node sends the output message on each terminal, but sends on the next terminal only when the processing has completed for the current terminal.

Updates to a message are never propagated to nodes which have been previously executed, only to nodes that follow the node in which the update has been made. The order in which the message is propagated to the different output terminals is determined by the broker; you cannot change this order. The only exception to this rule is the FlowOrder node, in which the terminals indicate the order in which the message is propagated to each.

All built-in nodes include error handling as part of their processing. If an error is detected within the node, the message is propagated to the failure terminal. What happens then depends on the structure of your message flow. You can use only the basic error handling provided by the broker, or you can enhance your flow by adding error processing nodes and flows to provide more comprehensive failure processing. For more information about these options, see "Handling errors in message flows" on page 244.

The message flow can accept a new message for processing only when all paths through the message flow (that is, all connected nodes from all output terminals) have been completed, and control has returned to the input node which commits or rolls back the transaction.

The following sample uses environment variables in the XML_Reservation sample to store information that has been taken from a database table and to pass that information to a node downstream in the message flow.

- Airline Reservations

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

Message flow node palette

The palette in the Message Flow editor contains all the built-in nodes, which are organized into categories, or drawers. A drawer is a container for a list of icons, such as the **Favorites** drawer.

When you first open the workbench, the default drawers contain built-in nodes that are related in function. For example, one drawer contains all the nodes that

handle input and output to WebSphere MQ queues. Another drawer, **Transformation**, groups the nodes that you can use to convert the input message into a different form, including the Compute and JavaCompute nodes.

You can drag the nodes that you use most often into the **Favorites** drawer for easy access. If you create your own nodes, you can also add them to the palette. You can drag a node from the palette onto the canvas, and create a connection between two nodes.

Right-click the palette to add a selected node to the canvas, or customize the appearance and behavior of the palette.

Use the Customize Palette dialog box to reorder node categories, set the drawer behavior for individual categories, and rename or hide nodes or categories.

You cannot move a category above the Favorites category. You can hide the Favorites category, but you cannot delete or rename it.

Use the Palette Settings dialog box to set the palette layout, determine the behavior of palette drawers, and choose a particular font.

The following topics explain how to change the palette layout and settings:

- "Changing the palette layout" on page 269
- "Changing the palette settings" on page 270
- "Customizing the palette" on page 270

Message flows, ESQL, and mappings

A message flow represents the set of actions that are performed on a message when it is received and processed by a broker.

The content and behavior of a message flow is defined by a set of files that you create when you complete your definition and configuration of the message flow content and structure:

- The message flow definition file `<message_flow_name>.msgflow`. This file is mandatory, and is created automatically for you. It contains details about the message flow characteristics and contents (for example, what nodes it includes, its promoted properties, and so on).
- The ESQL resources file `<message_flow_name>.esql`. This file is required only if your message flow includes one or more of the nodes that you can customize by using ESQL modules. You can create this file yourself, or you can cause it to be created for you by requesting specific actions against a node.

  You can customize the following built-in nodes by creating free-form ESQL statements that use the built-in ESQL statements and functions, and your own user-defined functions:
  - Compute
  - Database
  - Filter
- The message mappings file `<message_flow_name><_nodename>.msgmap`. This file is required only if your message flow contains one or more of the nodes that you can customize by using mappings. You can create this file yourself, or you can cause it to be created for you by requesting specific actions against a node. A different file is required for each node in the message flow that uses the Message Mapping editor.

You can customize the following built-in nodes by specifying how input values map to output values.

Node	Usage
"DataDelete node" on page 923	Use this node to delete one or more rows from a database table without creating an output message.
"DataInsert node" on page 926	Use this node to insert one or more rows in a database table without creating an output message.
"DataUpdate node" on page 930	Use this node to update one or more rows in a database table without creating an output message.
"Mapping node" on page 1052	Use this node to construct output messages and populate them with information that is new, modified from the input message, or taken from a database. You can also use the Mapping node to update, insert, or delete rows in a database table.
"Warehouse node" on page 1307	Use this node to store all or part of a message in a database table without creating an output message.

You can use built-in ESQL functions and statements to define message mappings, and you can use your own ESQL functions.

The "Extract node" on page 944 also uses mappings to create a new output message that contains a subset of the contents of the input message. However, the Extract node is deprecated in WebSphere Message Broker Version 6.0 and later releases. Although message flows that contain an Extract node remain valid in WebSphere Message Broker Version 6.0, redesign your message flows to replace Extract nodes by Mapping nodes to take advantage of later enhancements.

WebSphere Adapters nodes

A WebSphere Adapters node is a message flow node that is used to communicate with enterprise information systems (EIS), such as SAP, Siebel, and PeopleSoft.

The following terms are associated with WebSphere Adapters:

EIS Enterprise information system. This term is used to describe the applications that form an enterprise's existing system for handling company-wide information. An enterprise information system offers a well-defined set of services that are exposed as local or remote interfaces or both. Enterprise Resource Planning (ERP) and Customer Relationship Management (CRM) are typical enterprise information systems.

EMD Enterprise Metadata Discovery. A specification that you can use to examine an EIS and get details of business object data structures and APIs. An EMD stores definitions as XML schemas by default, and builds components that can access the EIS. In WebSphere Message Broker you use the Adapter Connection wizard to examine an EIS.

Business object
In a development or production environment, a set of XML schema attributes that represents a business entity (such as an invoice) and the definition of actions that can be performed on those attributes (such as the create and update operations).

The WebSphere Adapters support two modes of communication:

- **Inbound**: An event is generated on the EIS and the adapter responds to the event by sending a message to the message broker. The WebSphere Adapters input nodes support inbound communication. When the EIS sends an event to the adapter, a message is propagated from the WebSphere Adapters input node. For example, use an SAPInput node to accept input from an SAP application.
- **Outbound**: The message broker uses the adapter to send a request to the EIS. The WebSphere Adapters request nodes support outbound communication. When a message is propagated to the WebSphere Adapters request node, the adapter sends a request to the EIS. For example, use an SAPRequest node to send requests to an SAP application.

The WebSphere Adapters nodes need an adapter component to access the EIS. The input nodes need an inbound adapter component, which allows the EIS to invoke the message flow when an event occurs. The request nodes need an outbound adapter component, which is used by the message flow to invoke a service in the EIS.

The WebSphere Adapters nodes also need a message set to ensure that the WebSphere Message Broker messages that are propagated to and from the nodes reflect the logical structure of the data in the EIS.

SAP, Siebel, and PeopleSoft adapters are supported by the following message flow nodes in WebSphere Message Broker:
- SAPInput node
- SAPRequest node
- SiebelInput node
- SiebelRequest node
- PeopleSoftInput node
- PeopleSoftRequest node

The TwineballInput and TwineballRequest nodes are sample nodes with their own sample EIS. You can use the Twineball nodes to see how adapters nodes work. You cannot use the Twineball nodes to connect to the external SAP, Siebel, and PeopleSoft EIS systems.

You can configure the Websphere Adapters nodes by using properties on the nodes, or by using a configurable service. For example, you can use a configurable service to specify the connection details for the EIS. For more information, see Configurable services properties.

The mode in which your broker is working can affect the number of execution groups and message flows that you can deploy, and the type of node that you can use. For example, in Remote Adapter Deployment mode, only adapter-related features are enabled, and the types of node that you can use, and the number of execution groups that you can create, are limited. For more information about the available modes of operation, see Operation modes.

For more information about support for adapters on different operating systems, see WebSphere Message Broker Requirements.

The following topics provide an overview of the WebSphere Adapters:
- "Overview of WebSphere Adapter for SAP Software" on page 12
- "Overview of WebSphere Adapter for Siebel Business Applications" on page 48
- "Overview of WebSphere Adapter for PeopleSoft Enterprise" on page 54

Overview of WebSphere Adapter for SAP Software

With WebSphere Adapter for SAP Software you can create integrated processes that include the exchange of information with an SAP server, without special coding.

By using the adapter, an application component (the program or piece of code that performs a specific business function) can send requests to the SAP server (for example, to query a customer record in an SAP table or to update an order document) or receive events from the server (for example, to be notified that a customer record has been updated). The adapter creates a standard interface to the applications and data on the SAP server so that the developer of the application component does not have to understand the lower-level details (the implementation of the application or the data structures) on the SAP server.

WebSphere Adapter for SAP Software complies with the Java Connector Architecture (JCA) 1.5, which standardizes the way in which application components, application servers, and Enterprise Information Systems (EIS), such as an SAP server, interact with each other.

The adapter, which you generate with the Adapter Connection wizard, uses a standard interface and standard data objects. The adapter takes the standard data object sent by the application component and calls the SAP function. The adapter then returns a standard data object to the application component. The application component does not have to deal directly with the SAP function; it is the SAP adapter that calls the function and returns the results.

For example, the application component that requested the list of customers sends a standard business object with the range of customer IDs to the SAP adapter. The application component receives, in return, the results (the list of customers) in the form of a standard business object. The adapter completes all the interactions directly with the SAP function.

Similarly, the message flow might want to know about a change to the data on the SAP server (for example, a change to a particular customer). You can generate an adapter component that listens for such events on the SAP server and notifies message flows with the update. In this case, the interaction begins at the SAP server.

For more information, see “Technical overview of Adapter for SAP Software.”

Technical overview of Adapter for SAP Software:

WebSphere Adapter for SAP Software provides multiple ways to interact with applications and data on SAP servers. Outbound processing (from an application to the adapter to the SAP server) and inbound processing (from the SAP server to the adapter to an application) are supported.

WebSphere Adapter for SAP Software connects to SAP systems running on SAP Web application servers. The adapter supports Advanced Event Processing (AEP) and Application Link Enabling (ALE) for inbound processing, and the Business Application Programming Interface (BAPI), AEP, ALE, and Query Interface for SAP Systems (QISS) for outbound processing. You set up the adapter to perform outbound and inbound processing by using the Adapter Connection wizard to generate business objects based on the services it discovers on the SAP server.

For outbound processing, the adapter client invokes the adapter operation to create, update, or delete data on the SAP server or to retrieve data from the SAP server.

For inbound processing, an event that occurs on the SAP server is sent from the SAP server to the adapter. The ALE inbound and BAPI inbound interfaces start event listeners that detect the events. Conversely, the Advanced event processing interface polls the SAP server for events. The adapter then delivers the event to an endpoint, which is an application or other consumer of the event from the SAP server.

You configure the adapter to perform outbound and inbound processing by using the Adapter Connection wizard to create a message set that includes the interface to the SAP application as well as business objects based on the functions or tables that it discovers on the SAP server.

Overview of the outbound processing interfaces

WebSphere Adapter for SAP Software provides multiple interfaces to the SAP server for outbound processing.

- Through its BAPI interfaces, the adapter issues remote function calls (RFCs) to RFC-enabled functions, such as a Business Application Programming Interface (BAPI) function. These remote function calls create, update, or retrieve data on an SAP server.
  - The BAPI interface works with individual BAPIs (simple BAPIs). For example, you might want to check to see whether specific customer information exists in an SAP database.
  - The BAPI work unit interface works with ordered sets of BAPIs. For example, you might want to update an employee record. To do so, you use three BAPIs:
    1. To lock the record (to prevent any other changes to the record)
    2. To update the record
    3. To have the record approved
  - The BAPI result set interface uses two BAPIs to select multiple rows of data from an SAP database.

  BAPI calls are useful when you need to perform data retrieval or manipulation and a BAPI or RFC function that performs the task already exists.

  Simple BAPIs can be sent through the synchronous RFC, asynchronous transactional RFC, or asynchronous queued RFC protocol.
  - With synchronous RFC, both the adapter and the SAP server must be available when the call is made from the adapter to the SAP server. The adapter sends a request to the SAP server and waits for a response.
  - With asynchronous transactional RFC, a transaction ID is associated with the call from the adapter to the SAP server. The adapter does not wait for a response from the SAP server. Only the transaction ID is returned to the message flow.
  - With asynchronous queued RFC, the call from the adapter is delivered to a predefined queue on the SAP server. As with asynchronous RFC, a transaction ID is associated with the call, and the adapter does not wait for a response from the SAP server.

    This interface is useful when the event sequence must be preserved.
- The Query interface for SAP Software retrieves data from specific SAP application tables. It can return the data or check for the existence of the data.

You can use this type of interaction with SAP if you need to retrieve data from an SAP table without using an RFC function or a BAPI.

- With the Application Link Enabling (ALE) interface, you exchange data using SAP Intermediate Data structures (IDocs). For outbound processing, you send an IDoc or a packet of IDocs to the SAP server.

  The ALE interface, which is particularly useful for batch processing of IDocs, provides asynchronous exchange. You can use the queued transactional (qRFC) protocol to send the IDocs to a queue on the SAP server. The qRFC protocol ensures the order in which the IDocs are received. It is often used for system replications or system-to-system transfers.
- With the ALE pass-through IDoc interface, the adapter sends the IDoc to the SAP server with no conversion of the IDoc. The Message tree contains a BLOB field that represents the IDoc.
- With the Advanced event processing interface, you send data to the SAP server. The data is then processed by an ABAP handler on the SAP server.

Overview of the inbound processing interfaces

WebSphere Adapter for SAP Software provides the following interfaces to the SAP server for inbound processing.

- Through its BAPI inbound interface, the adapter listens for events and receives notifications of RFC-enabled function calls from the SAP server.
  - With synchronous RFC, both the adapter and the SAP server must be available when the call is made from the SAP server to the adapter. The adapter sends the request to a predefined application and returns the response to the SAP server.
  - With asynchronous transactional RFC, the event is delivered to the adapter even if the adapter is not available when the call is made. The SAP server stores the event on a list of functions to be invoked and continues to attempt to deliver it until the adapter is available.

    You also use asynchronous transaction RFC if you want to deliver the functions from a predefined queue on the SAP server. Delivering the files from a queue ensures the order in which the functions are sent.

    If you select assured once-only delivery, the adapter uses a data source to persist the event data received from the SAP server. Event recovery is provided to track and recover events in case a problem occurs when the adapter attempts to deliver the event to the endpoint.
- With the ALE inbound processing interface, the adapter listens for events and receives one or more IDocs from the SAP server. As with ALE outbound processing, ALE inbound processing provides asynchronous exchange.

  You can use the qRFC interface to receive the IDocs from a queue on the SAP server, which ensures the order in which the IDocs are received.

  If you select assured once-only delivery, the adapter uses a data source to persist the event data, and event recovery is provided to track and recover events in case a problem occurs when the adapter attempts to deliver the event to the endpoint.
- With the ALE pass-through IDoc interface, the SAP server sends the IDoc through the adapter to the endpoint with no conversion of the IDoc. The Message tree contains a BLOB field that represents the IDoc.

- The Advanced event processing interface polls the SAP server for events. It discovers events waiting to be processed. It then processes the events and sends them to the endpoint. For more information, see "The Advanced event processing interface" on page 41.

How the adapter interacts with the SAP server

The adapter uses the SAP Java Connector (SAP JCo) API to communicate with SAP applications. An application sends a request to the adapter, which uses the SAP JCo API to convert the request into a BAPI function call. The SAP system processes the request and sends the results to the adapter. The adapter sends the results in a response message to the calling application.

For more information, see the following topics.

- "The Adapter Connection wizard (SAP)"
- "The BAPI interfaces"
- "The ALE interfaces" on page 28
- "Query interface for SAP Software" on page 38
- "The Advanced event processing interface" on page 41

The Adapter Connection wizard (SAP):

The Adapter Connection wizard is a tool that you use to create services. The Adapter Connection wizard establishes a connection to the SAP server, discovers services (based on search criteria that you provide), and generates business objects, interfaces, and import or export files, based on the services that are discovered.

By using WebSphere Message Broker, you establish a connection to the SAP server to browse the metadata repository on the SAP server. The SAP metadata repository, which is a database of the SAP data, provides a consistent and reliable means of access to that data.

You specify connection information (such as the user name and password needed to access the server), and you specify the interface that you want to use (for example, BAPI). The service metadata that is associated with that interface is displayed. You can then provide search criteria and select the information (for example, you can list all BAPIs that relate to "CUSTOMER" by using the search filter with "BAPI_CUSTOMER*", then select one or more BAPIs).

The result of running the Adapter Connection wizard is an adapter connection project and a message set project that contain the interfaces and business objects as well as the adapter.

The Adapter Connection wizard also produces an import file (for outbound processing) or an export file (for inbound processing).

- The import file contains the managed connection factory property settings that you provide in the wizard.
- The export file contains the activation specification property settings you provide in the wizard.

The BAPI interfaces:

The WebSphere Adapter for SAP Software supports outbound processing for simple BAPIs, BAPI units of work, and BAPI result sets. In outbound processing,

message flows call BAPIs and other RFC-enabled functions on the SAP server. The adapter supports inbound processing for simple BAPIs only. In inbound processing, the SAP server sends an RFC-enabled function (such as a BAPI function) through the adapter to an endpoint.

For example, you want to build a service that creates a new customer on the SAP server. You run the Adapter Connection wizard to discover the BAPI_CUSTOMER_CREATEFROMDATA function, and the wizard generates the business-object definition for BAPI_CUSTOMER_CREATEFROMDATA, as well as other Service Component Architecture (SCA) service resources. During BAPI outbound processing, the adapter receives the service request and converts the data into a BAPI invocation.

BAPI interface (simple BAPIs)

A simple BAPI performs a single operation, such as retrieving a list of customers. The adapter supports simple BAPI calls by representing each with a single business object schema.

Simple BAPIs can be used for outbound or inbound processing. You can specify synchronous RFC processing or asynchronous transactional RFC (tRFC) processing when you configure a module for a simple BAPI. In addition, for outbound processing, you can specify asynchronous queued RFC (qRFC) processing, in which BAPIs are delivered to a predefined queue on the SAP server.

- In synchronous RFC processing, the SAP server and the adapter must be available during processing.
  - In outbound processing, the message flow sends a request, then waits for a response from the SAP server.
  - In inbound processing, the SAP server sends a request through the adapter to an endpoint and waits for a response from the adapter.
- In asynchronous tRFC outbound processing, the adapter associates a transaction ID with the function call to the SAP server. The adapter does not wait for a response from the SAP server. If the delivery is unsuccessful, the message flow can use the SAP transaction ID (TID) to make the request again. The TID is a field in your message.
- In asynchronous tRFC inbound processing, the adapter does not have to be available when the SAP server runs the function call. The function call is placed on a list of functions to be invoked, and the call is attempted until it is successful.

  To send function calls from a user-defined outbound queue on the SAP server, you also specify asynchronous tRFC inbound processing.
- In asynchronous qRFC outbound processing, the process is similar to asynchronous tRFC outbound processing. A TID is associated with the function call, and the adapter does not wait for a response from the SAP server. In addition, the BAPIs are delivered to a predefined queue on the SAP server. By sending BAPIs to the predefined queue, you can ensure the order in which they are delivered.

BAPI work unit interface

A BAPI work unit consists of a set of BAPIs that are processed in sequence to complete a task. For example, to update an employee record in the SAP system, the record needs to be locked before being updated. This task is accomplished by

calling three BAPIs, in sequence, in the same work unit. The following three BAPIs illustrate the kind of sequence that forms such a unit of work:
- BAPI_ADDRESSEMP_REQUEST
- BAPI_ADDRESSEMP_CHANGE
- BAPI_ADDRESSEMP_APPROVE

The first BAPI locks the employee record, the second updates the record, and the third approves the update. The advantage of using a BAPI unit of work is that the message flow can request the employee record change with a single call, even though the work unit consists of three separate functions. In addition, if SAP requires that the BAPIs be processed in a specific sequence for the business flow to complete correctly, the work unit supports this sequence.

BAPI result set interface

BAPI result sets use the GetList and GetDetail functions to retrieve an array of data from the SAP server. The information that is returned from the GetList function is used as input to the GetDetail function.

For example, if you want to retrieve information on a set of customers, you use BAPI_CUSTOMER_GETLIST, which acts as the query BAPI, and BAPI_CUSTOMER_GETDETAIL, which acts as the result BAPI. The BAPIs perform the following steps:

1. The BAPI_CUSTOMER_GETLIST call returns a list of keys (for example, CustomerNumber).
2. Each key is mapped dynamically to the business object for BAPI_CUSTOMER_GETDETAIL.
3. BAPI_CUSTOMER_GETDETAIL is processed multiple times, so that an array of customer information is returned.

You use the Adapter Connection wizard to discover the BAPI_CUSTOMER_GETLIST and BAPI_CUSTOMER_GETDETAIL functions and build the key relationship between the two BAPIs. The wizard then generates business object definitions for these BAPIs as well as other SCA service resources. At run time, the client sets the values in the BAPI_CUSTOMER_GETLIST business object, and the adapter returns the corresponding set of customer detail records from the SAP server.

For more information, see the following topics.
- “Outbound processing for the BAPI interface”
- “Business objects for the BAPI interface” on page 26

Outbound processing for the BAPI interface:

In BAPI outbound processing, a message flow sends a request to the SAP server. For BAPI units of work and BAPI result sets, processing is handled synchronously (the message flow waits for a response from the SAP server). For simple BAPIs, you can request that processing be handled synchronously or asynchronously (the message flow does not wait for a response from the SAP server).

For BAPI units of work and BAPI result sets, the processing is handled as described in “Synchronous RFC” on page 18. For simple BAPIs, you make a selection, during configuration, about the type of remote RFC call you want to make.

Synchronous RFC

If you select **Synchronous RFC** (the default) during configuration for a simple BAPI, or if you are using BAPI units of work or BAPI result sets, the following processing steps occur:

1. The adapter receives a request from a message flow in the form of a BAPI business object.
2. The adapter converts the BAPI business object to an SAP JCo function call.
3. The adapter uses the Remote Function Call (RFC) interface to process the BAPI or RFC function call in the SAP application.
4. After passing the data to the SAP server, the adapter handles the response from SAP and converts it back into the business object format required by the message flow.
5. The adapter then sends the response back to the message flow.

Asynchronous transactional RFC

If you select **Asynchronous transactional RFC** during configuration, the following processing steps occur:

1. The adapter receives a request from a message flow in the form of a BAPI business object.
2. The adapter checks the business object to see whether the SAP transaction ID attribute has a value assigned. (The SAP transaction ID (TID) is a field in your message.)
   - If the SAP transaction ID attribute has a value, the adapter uses that value during processing.
   - If the attribute does not have a value, the adapter makes a call to the SAP server and gets a transaction ID from the SAP server.
3. The adapter converts the BAPI business object to an SAP JCo function call.
4. The adapter uses the transactional Remote Function Call (tRFC) protocol to make the call to the SAP server.

   The adapter does not wait for a response from the SAP server.
5. After the function data is passed to the SAP application, control returns to the adapter.
   - If the call to the SAP server fails, the SAP server throws an ABAPException.
   - If the call to the SAP server succeeds but contains invalid data, no exception is returned to the adapter. For example, if the adapter sends a request that contains an invalid customer number, the adapter does not respond with an exception indicating that no such customer exists.
6. The request node builds a message tree that contains the transaction ID as one of the fields.

Asynchronous queued RFC

If you select **Asynchronous queued RFC** during configuration, the following processing steps occur:

1. The adapter receives a request from a message flow in the form of a BAPI business object.
2. The adapter checks the business object to see whether the SAP transaction ID attribute has a value assigned. (The SAP transaction ID (TID) is a field in your message.)
   - If the SAP transaction ID attribute has a value, the adapter uses that value during processing.

- If the attribute does not have a value, the adapter makes a call to the SAP server and gets a transaction ID from the SAP server.

3. The adapter converts the BAPI business object to an SAP JCo function call.
4. The adapter uses the tRFC protocol to make the call to the specified queue on the SAP server.

   The adapter does not wait for a response from the SAP server.
5. After the function data is passed to the SAP application, control returns to the adapter.
   - If the call to the SAP server fails, the SAP server throws an ABAPException.
   - If the call to the SAP server succeeds but contains invalid data, no exception is returned to the adapter. For example, if the adapter sends a request that contains an invalid customer number, the adapter does not respond with an exception indicating that no such customer exists.
6. The request node builds a message tree that contains the transaction ID as one of the fields.

SAP BAPI transaction commit:

When the SAP adapter is used with the BAPI interface, you must consider certain factors when you design transactional flows.

You can configure message flows to be transactional so that updates to resources such as databases can be coordinated; changes are committed or rolled back together within the same transaction. This transactional coordination can be extended to external system updates, such as SAP databases, when you use the BAPI interface with SAPRequest nodes.

The SAP adapter can control whether it waits for SAP to commit the updates synchronously, or issues a commit and returns while the SAP commit happens asynchronously. You can determine this behavior by using the **Use wait parameter before calling BAPI commit** parameter on the Configure Objects pane of the Adapter Connection wizard. The adapter relies on the transactionality setting of the message flow to determine whether to issue the commit call.

BAPIs with implicit commit

In earlier releases of SAP, some BAPIs were coded with a commit. From SAP Release 4.0A onwards, it is more effective for BAPIs to issue a separate BAPI_TRANSACTION_COMMIT to force the update, instead of doing commit work. By using this method, BAPI calls can be made before the work is committed as a batched unit of work. To find out if a BAPI is coded with a commit, see the documentation for the BAPI.

Message flow transactionality

When the Transaction mode property on the SAPRequest node is set to Yes, the adapter is instructed to issue the SAP commit on completion of the message flow in line with other database commits. You can set the **Use wait parameter before calling BAPI commit** parameter in the Adapter Connection wizard that determines whether the commit is synchronous or asynchronous.

If the Transaction mode property on the SAPRequest node is set to No, the adapter does not issue an SAP commit and the parameter that you set on the Adapter Connection wizard has no relevance. However, the commit can still be issued as

part of a BAPI work unit COMMIT verb (to which the property on the wizard does apply) or a call to the BAPI_TRANSACTION_COMMIT (to which the property on the wizard does not apply).

The following rules apply when you set the Transaction mode property on the SAPRequest node.

- Set Transaction mode to No if the following conditions apply:
  - The BAPIs already have commits
  - A BAPI_TRANSACTION_COMMIT is called by an SAPRequest node
  - A BAPI work unit includes a BAPI_TRANSACTION_COMMIT or the COMMIT verb is added on the Configure Objects pane of the Adapter Connection wizard

  If the BAPIs are coded with commits and you set Transaction mode to Yes, the BAPI is called as part of the same transaction as those from other SAPRequest nodes in the same flow and using the same adapter. Therefore, any BAPIs that were called previously in this message flow are committed.
- Set Transaction mode to Yes if the following conditions apply:
  - The BAPI needs to be committed (that is, the BAPI is not coded with a commit)
  - The BAPI work unit needs to be committed and does not include a BAPI_TRANSACTION_COMMIT or the COMMIT verb

  If you set Transaction mode to No, the BAPI is not committed now or at the end of the message flow; it is not guaranteed ever to be committed.

The following scenarios illustrate the visibility of the updates made to an SAP system, and show how to use the adapter to avoid uncertainty when data is being committed by an external system.

- Scenario 1: Business partner and relationship processing in a single flow
- Scenario 2: Order create and query application processing with two flows

Scenario 1: Business partner and relationship processing in a single flow:

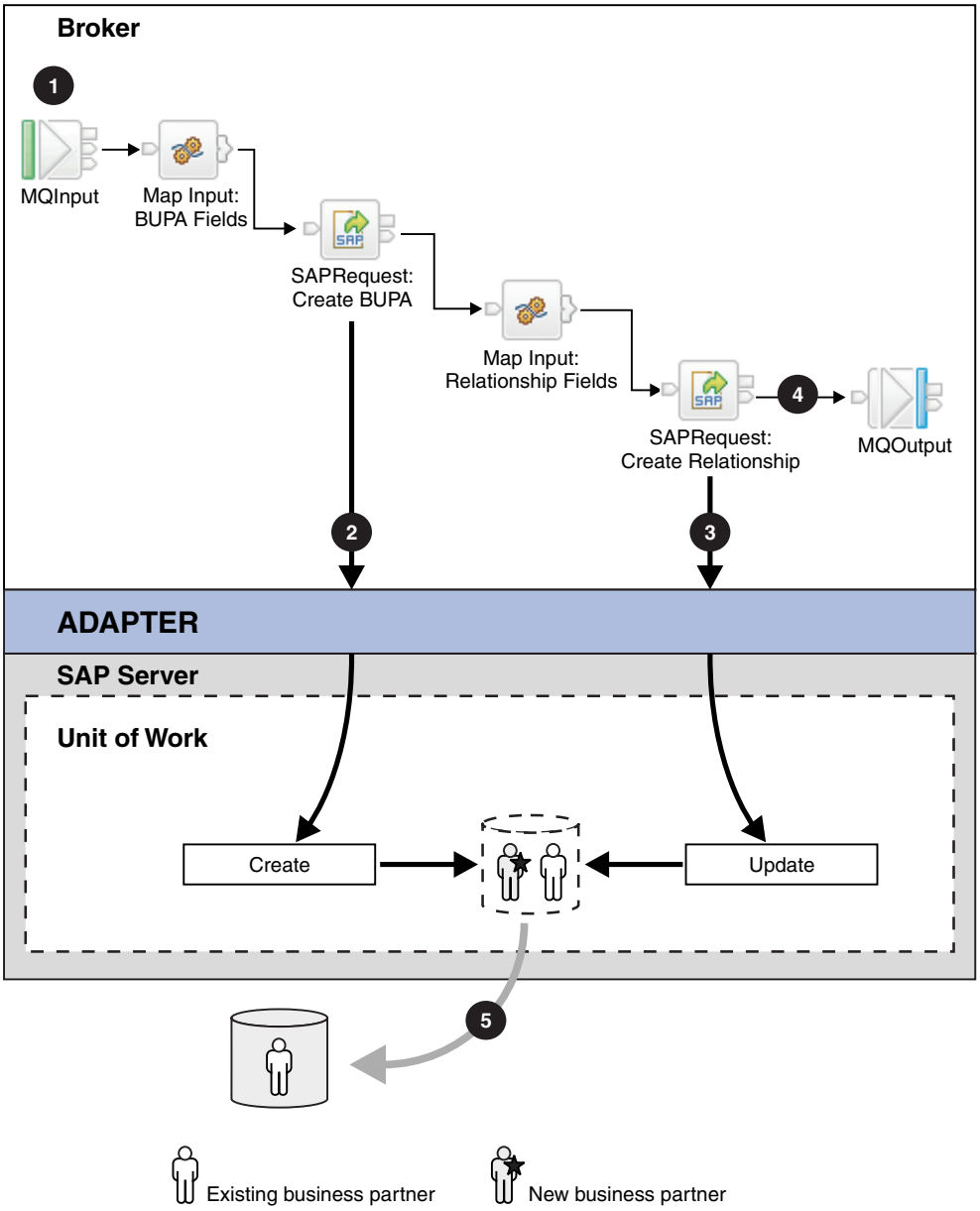
You must set the Transaction mode property appropriately on an SAPRequest node when you are processing in a single message flow.

This scenario is one of two examples that illustrate the concepts that are described in “SAP BAPI transaction commit” on page 19; see also “Scenario 2: Order create and query application processing with two flows” on page 22.

In this scenario, a message flow is used to create a business partner, and a new relationship with an existing partner by using two BAPI calls:

```
BAPI_BUPA_CREATE_FROM_DATA
BAPI_BUPR_RELATIONSHIP_CREATE
```

The message flow consists of two SAPRequest nodes with the Transaction mode property set to Yes on both nodes to allow commit or rollback in case of exceptions. When the Transaction mode property is set to Yes, the message flow's final commit takes place at the end of the flow when the adapter requests SAP to commit the order.



1. An application triggers the transactional flow that creates the business partner.
2. The SAPRequest node submits a BUPA creation and returns the business partner number. The commit happens when the message flow completes because the node participates in a message flow level transaction.
3. The second SAPRequest node attempts to create a relationship between an existing business partner and the new business partner; however, SAP has not yet committed the creation of the new business partner to the database.

   If the same adapter is used for both BAPIs, the adapter ensures a single connection to SAP because both nodes have to participate in the same logical work unit. The single connection means that the BUPA creation is visible to the relationship update call (3 in the diagram), even though the flow transactionality has yet to initiate the commit.

   If the Transaction mode property were set to Yes on the create BUPA call, but No on the create relationship call, the adapter would need to use two different connections to SAP; that is, the transactional properties of the connections

would be different. The create relationship call would therefore fail because the new business partner would not be visible until the message flow and the transactional commit have completed.

4. The MQOutput node puts an MQ message on the output queue pending transactional commit.
5. The message flow completes and the broker begins to commit all the resources involved in that flow, including SAP (5 in the diagram). The updates are committed in SAP.

This scenario illustrates the ability of the broker to use its transactional control of the message flow to provide the necessary information to the SAPRequest nodes to carry out related processing, even though the external SAP system is committing the work asynchronously.

Scenario 2: Order create and query application processing with two flows:

You must set the Transaction mode property appropriately on an SAPRequest node when you are processing by using separate message flows.

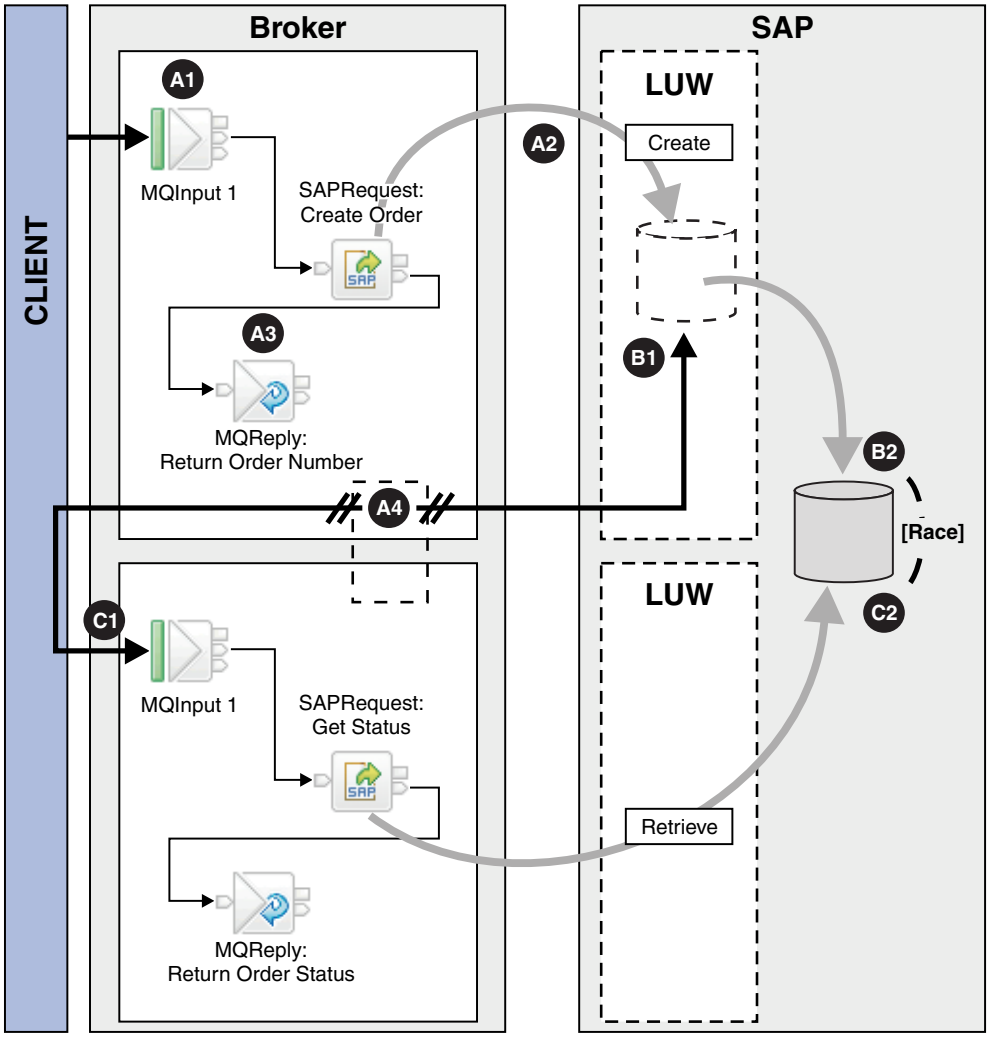
This scenario is one of two examples that illustrate the concepts that are described in "SAP BAPI transaction commit" on page 19; see also "Scenario 1: Business partner and relationship processing in a single flow" on page 20.

In this scenario, two message flows are used to issue a sales order create and a subsequent sales order check by using two BAPI calls:

```
BAPI_SALESORDER_CREATEFROMDAT2
BAPI_SALESORDER_GETSTATUS
```

For example, a user queries an order after a purchase has been made by using a Web-based application. The result of the query is closely linked to the asynchronous or synchronous behavior of the order creation steps carried out by the external SAP server.

In the following asynchronous example (the default behavior), the query might fail and the user receives a negative acknowledgment for the order that has been created.



BAPI Create Order message flow

A1. An application triggers the transactional message flow that creates a sales order.

A2. The SAPRequest node submits an order creation and returns the order registration number. The commit happens when the message flow completes because the node participates in a message flow level transaction.

A3. The MQReply node puts an MQ message on the output queue pending transactional commit.

A4. The message flow completes and the broker begins to commit all the resources involved in that flow, including SAP and the MQReply node call. The order number is available to the user application.

The following two processes occur simultaneously, and effectively race each other to complete.

SAP Commits processing asynchronously

B1. The SAP commit begins.

BAPI Get Order Status message flow

C1. A request for an order status query is made.

B2. The SAP commit completes.

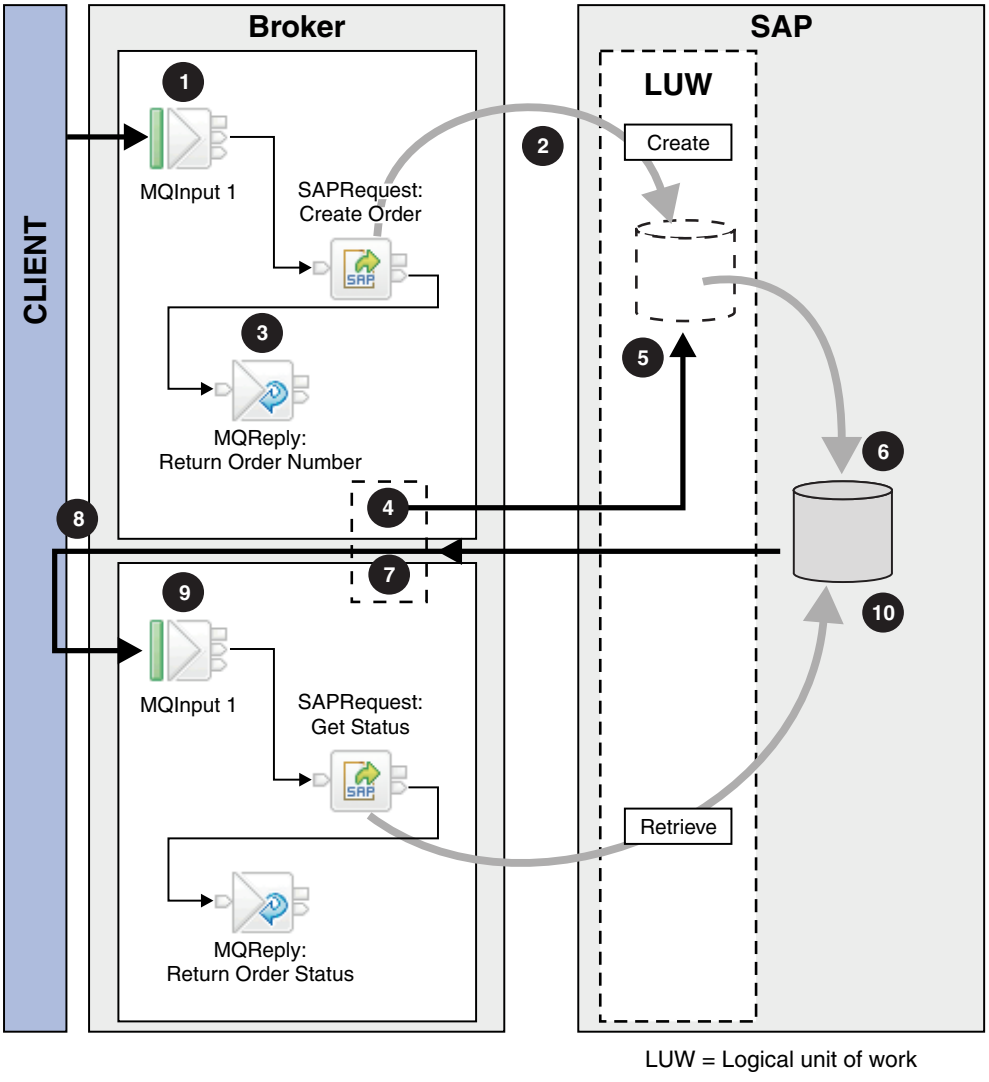
C2. The SAPRequest node requests the order.

Because of the asynchronous commit, two outcomes are possible when the order is queried:

- The order is not found because SAP has not completed the order commit.
- The order is found but only if SAP has committed the order before the query is made.

You can avoid this uncertainty by configuring the adapter to perform the commit synchronously; set the **Use wait parameter before calling BAPI commit** parameter on the adapter connection wizard to True, and set the Transaction mode property on the SAPRequest node to Yes.

In the following synchronous example, the query is successful and the user receives a positive acknowledgment for the order that has been created.



BAPI Create Order message flow

1. An application triggers the transactional message flow that creates the sales order.
2. The SAPRequest node submits an order creation and returns the order registration number. The commit happens when the message flow completes because the node participates in the a message flow level transaction.
3. The MQReply node puts an MQ message on the output queue pending transactional commit.
4. The message flow completes and the broker begins to commit all the resources involved in that flow, including SAP.

SAP Commits processing synchronously

5. The SAP commit begins.
6. The SAP commit completes.
7. The adapter hands control back to the broker.
8. The MQReply node call is committed, therefore the order number is available to the user application.

BAPI Get Order Status message flow

9. A request for an order status query is made.
10. The SAPRequest node requests the order.

SAP has completed the order commit; therefore, the order query is successful.

Inbound processing for the BAPI interface:

The adapter supports inbound processing (from the SAP server to the adapter) for simple BAPIs.

A client application on the SAP server invokes a function through the adapter to an end point.

For more information, see the following topics:
- "Synchronous and asynchronous RFC"
- "Event recovery for the BAPI interface" on page 26

Synchronous and asynchronous RFC:

For BAPI outbound processing, you can specify that the processing be handled synchronously or asynchronously. However, for BAPI inbound processing in WebSphere Message Broker, you can specify that the processing be handled asynchronously only. In asynchronous processing, the SAP application does not wait for a response and the adapter does not have to be available when the SAP application makes the function call.

The BAPI interface has a set of activation specification properties for asynchronous RFC, which you use to set up inbound processing. You specify values for the properties with the Adapter Connection wizard.

The sequence of processing actions that result from an inbound request differ, depending on the selection you make during configuration from the **SAP Remote Function Call (RFC) type** list.

Asynchronous transactional RFC

If you select **Asynchronous Transactional/Queued RFC** during configuration, the following processing steps occur:

1. A client on the SAP server invokes an RFC-enabled function call on the adapter.

Note: To send the RFC-enabled functions from a queue on the SAP server, the client program on the SAP server delivers the events to a user-defined outbound queue.

A transaction ID is associated with the call.

The calling program on the SAP server does not wait to see whether the call to the adapter was successful, and no data is returned to the calling program.

2. The RFC-function call is placed on a list of functions to be delivered.

   You can see the event list by entering transaction code SM58 on the SAP server
3. The RFC-function call is invoked on the adapter. If the adapter is not available, the call remains in the list on the SAP server, and the call is repeated at regular intervals until it can be processed by the adapter.

   When the SAP server successfully delivers the call event, it removes the function from the list.
4. If you selected **Ensure once-only event delivery**, the adapter sets the transaction ID in the event persistent table.

   This is to ensure the event is not processed more than once.
5. The adapter resolves the operation and business object name using the received RFC-enabled function name.
6. The adapter sends the business object to an endpoint.

   If you are sending functions from a user-defined queue on the SAP server, the functions are delivered in the order in which they exist on the queue. You can see the contents of the queue by entering transaction code SMQ1 on the SAP server.
7. If the delivery is successful, and if you selected **Ensure once-only event delivery**, the adapter removes the transaction ID from the event persistent table.

   If a failure occurs when the adapter attempts to deliver the business object, the transaction ID remains in the event table. When another event is received from the SAP server, the following processing occurs:
   a. The adapter checks the transaction ID.
   b. If the event matches an ID in the table, the adapter processes the failed event once; it does not send the event with the duplicate ID, thereby ensuring that the event is processed only once.

Event recovery for the BAPI interface:

You can configure the adapter for BAPI inbound processing so that it supports event recovery in case a failure occurs while the event is being delivered from the adapter to the endpoint. When event recovery is specified, the adapter persists the event state in an event recovery table that resides on a data source.

Event recovery is not the default; you must specify it by enabling once-only delivery of events during adapter configuration.

Business objects for the BAPI interface:

A business object is a structure that consists of data, the action to be performed on the data, and additional instructions for processing the data.

For outbound processing, the broker uses business objects to send data to SAP or obtain data (through the adapter) from SAP. The broker sends a business object to the adapter, and the adapter converts the data in the business object to a format that is compatible with an SAP API call. The adapter then runs the SAP API with this data.

For inbound processing, the SAP server sends a BAPI function call through the adapter to an endpoint. The adapter converts the BAPI function call into a business object.

The adapter uses the BAPI metadata that is generated by the Adapter Connection wizard to construct a business-object definition. This metadata contains BAPI-related information such as the operation of the business object, import parameters, export parameters, table parameters, transaction information, and dependent or grouped BAPIs.

The BAPI business-object definition that is generated by the Adapter Connection wizard is modeled on the BAPI function interface in SAP. The business-object definition represents a BAPI function, such as a BAPI_CUSTOMER_GETLIST function call.

If you change the BAPI interface, you must run the Adapter Connection wizard again to rebuild the business object definition.

How business-object definitions are created

You create business-object definitions by using the Adapter Connection wizard. The wizard connects to the application, discovers data structures in the application, and generates business-object definitions to represent them. It also generates other resources that are needed by the adapter, such as the interface information that indicates the input and output parameters.

Business object structure

The structure of a BAPI business object depends on the interface type (simple BAPI, BAPI work unit, or BAPI result set).

For more information, see the following topics.

- "Business object structure for a simple BAPI"
- "Business object structure for a nested BAPI" on page 28
- "Business object structure for a BAPI work unit" on page 28
- "Business object structure for a BAPI result set" on page 28

Business object structure for a simple BAPI:

A business object for a simple BAPI call reflects a BAPI method or function call in SAP. Each business object property maps to a BAPI parameter. The metadata of each business-object property indicates the corresponding BAPI parameter. The operation metadata determines the correct BAPI to call.

For a simple BAPI that performs Create, Update, Retrieve, and Delete operations, each operation is represented by a business object, with the business objects being grouped together in a wrapper.

The business object wrapper can be associated with multiple operations, but for a simple BAPI, each business object is associated with only one operation. For example, while a wrapper business object can contain BAPIs for Create and Delete operations, BAPI_CUSTOMER_CREATE is associated with the Create operation, not the Delete operation.

The BAPI business objects are children of the business object wrapper, and, depending on the operation to be performed, only one child object in this wrapper needs to be populated at run time in order to process the simple BAPI call. Only one BAPI, the one that is associated with the operation to be performed, is called at a time.

If you select **Asynchronous Transactional RFC** (for outbound or inbound processing) or **Asynchronous Queued RFC** (for outbound processing) , the BAPI wrapper business object also contains a transaction ID. The transaction ID is used to resend the BAPI call if the receiving system is not available at the time of the initial call.

Business object structure for a nested BAPI:

A nested BAPI business object contains structure parameters that can contain one or more other structures as components.

A BAPI business object can contain both simple parameters and structure parameters. A business object that contains structure parameters can in turn contain other structures, such as simple parameters and a business object.

Business object structure for a BAPI work unit:

A business object that represents a BAPI work unit (also known as a BAPI transaction) is actually a wrapper object that contains multiple child BAPI objects. Each child BAPI object in the wrapper object represents a simple BAPI.

The adapter supports a BAPI work unit by using a top-level wrapper business object that consists of multiple child BAPIs, each one representing a simple BAPI in the sequence. The BAPI wrapper object represents the complete work unit, while the child BAPI objects contained in the BAPI wrapper object represent the individual operations that make up the work unit.

Business object structure for a BAPI result set:

The top-level business object for a result set is a wrapper that contains a GetDetail business object. The GetDetail business object contains the results of a query for SAP data. The GetDetail business object also contains, as a child object, the query business object. The query business object represents a GetList BAPI. These two BAPIs work together to retrieve information from the SAP server.

The ALE interfaces:

The SAP Application Link Enabling (ALE) interface and ALE pass-through IDoc interface enable business process integration and asynchronous data communication between two or more SAP systems or between SAP and external systems. The data is exchanged in the form of Intermediate Documents (IDocs).

The adapter supports outbound and inbound processing by enabling the exchange of data in the form of business objects.

- For inbound processing, SAP pushes the data in IDocs to the SAP adapter. The adapter converts the IDocs to business objects and delivers them to the endpoint.
- For outbound processing, the SAP adapter converts the business object to an IDoc and delivers it to SAP.

To use the ALE interface or ALE pass-through IDoc interface for inbound processing, make sure that your SAP server is properly configured (for example, you must set up a partner profile and register an SAP RCF program ID to listen for events).

Application systems are loosely coupled in an ALE integrated system, and the data is exchanged asynchronously.

IDocs

Intermediate Documents (IDocs) are containers for exchanging data in a predefined (structured ASCII) format across system boundaries. The IDoc type indicates the SAP format that is to be used to transfer the data. An IDoc type can transfer several message types (the logical messages that correspond to different business processes). IDocs can be used for outbound and inbound processing.

For example, if an application developer wants to be notified when a sales order is created on the SAP server, the developer runs the Adapter Connection wizard to discover the ORDERS05 IDoc on the SAP server. The wizard then generates the business object definition for ORDERS05. The wizard also generates other resources, such as an EIS export component and Service Component Architecture (SCA) interfaces.

IDocs are exchanged for inbound and outbound events, and IDocs can be exchanged either as individual documents or in packets.

The processing of IDoc data depends on whether you are using the ALE interface or the ALE pass-through IDoc interface.

- **ALE interface**

  For outbound processing, the adapter uses the IDoc business object to populate the appropriate RFC-enabled function call made to the SAP server.

  For inbound processing, IDocs can be sent from the SAP server as parsed or unparsed documents

  – For parsed documents, the adapter parses the IDoc and creates a business object that reflects the internal structure of the IDoc.

  – For unparsed IDocs, the adapter processes the IDoc but does not convert the data portion of the IDoc.

- **ALE pass-through IDoc interface**

  For both outbound and inbound processing, the adapter does no conversion of the IDoc, which is useful when the client wants to perform the IDoc parsing.

Transactional RFC processing

The adapter uses transactional RFC (tRFC) to assure delivery and to ensure that each IDoc is exchanged only once with SAP. The tRFC component stores the called RFC function in the database of the SAP system with a unique transaction identifier (TID). The TID is a field in your message.

The message flow must determine how to store the SAP transaction ID and how to relate the SAP transaction ID to the data being sent to the adapter. When the events are successful, the message flow should not resubmit the event associated with this TID again to prevent the processing of duplicate events.

- If the message flow does not send an SAP transaction ID with the business object, the adapter returns one after running the transaction.

- If the message flow has an SAP transaction ID, it must populate the SAP transaction ID property in the business object with that value before running the transaction.

The SAP transaction ID can be used for cross-referencing with a global unique ID that is created for an outbound event. You can create the global unique ID for managing integration scenarios.

Queued RFC processing

The adapter uses qRFC (queued transactional RFC) to ensure that IDocs are delivered in sequence to a queue on the SAP server or are received in sequence from the SAP server. Additional threads can increase the throughput of a message flow but you should consider the potential effect on message order. To maintain message order, ensure that your message flow is single threaded.

For more information about ALE interfaces, see the following topics:
- "Outbound processing for the ALE interface"
- "Inbound processing for the ALE interface" on page 31
- "Pass-through support for IDocs, and MQSeries link for R/3 link migration" on page 36
- "ALE business objects" on page 37

Outbound processing for the ALE interface:

The adapter supports outbound processing (from the adapter to the SAP server) for the ALE interface and the ALE pass-through IDoc interface. ALE uses IDocs for data exchange, and the adapter uses business objects to represent the IDocs.

The following list describes the sequence of processing actions that result from an outbound request that uses the ALE interface and ALE pass-through IDoc interface.

The message flow that makes the request uses the interface information that was generated by the Adapter Connection wizard.

1. The adapter receives a request, which includes an IDoc business object, from a message flow.

   For pass-through IDocs, the Message tree contains a BLOB field that represents the IDoc. No separate IDoc business object exists for pass-through IDocs.
2. The adapter uses the IDoc business object to populate the appropriate RFC-enabled function call used by the ALE interface.
3. The adapter establishes an RFC connection to the ALE interface and passes the IDoc data to the SAP system. If you are using the qRFC protocol, the adapter passes the IDoc data in the order specified in the wrapper business object to the specified queue on the SAP server.
4. After passing the data to SAP, the adapter performs one of the following steps:
   - If the call is not managed by a local transaction that uses the broker's Local Transaction Manager, the adapter releases the connection to SAP and does not return any data to the caller. When no exceptions are raised, the outbound transaction is considered successful. You can verify whether the data is incorporated into the SAP application by inspecting the IDocs that have been generated in SAP.
   - If the call is managed by a local transaction that uses the broker's Local Transaction Manager, the adapter returns the transaction ID.

The adapter uses the tRFC protocol to support J2C local transactions.

Inbound processing for the ALE interface:

The adapter supports inbound processing (from the SAP server to the adapter) for the ALE interface and the ALE pass-through IDoc interface.

When you are configuring a module for the ALE interface or the ALE pass-through interface, you indicate whether the IDocs are sent as a packet and, for the ALE interface, you can specify whether they are sent parsed or unparsed. You make these selections in the Adapter Connection wizard. When you use the ALE pass-through IDoc interface, the Message tree contains a BLOB field that represents the IDoc. No separate IDoc business object exists for pass-through IDocs.

The following list describes the sequence of processing actions that result from an inbound request using the ALE interface.

1. The adapter starts event listeners to the SAP server.
2. Whenever an event occurs in SAP, the event is sent to the adapter through the event listeners.
3. The adapter converts the event into a business object before sending it to the endpoint.

The adapter uses the event recovery mechanism to track and recover events in case of abrupt termination. The event recovery mechanism uses a data source for persisting the event state.

The following table provides an overview of the differences between the ALE interface and the ALE pass-through IDoc interface for inbound processing.

Interface	When to use	SplitIDoc = true	SplitIDoc = false	Parsed IDoc = true
ALE inbound	This interface converts the raw incoming IDocs to business objects, which are readily consumable by the client at the endpoint.	On receiving the IDoc packet from SAP, the adapter converts the IDocs to business objects, one by one, before sending each one to the endpoint.	On receiving the IDoc packet from SAP, the adapter converts the IDocs in the packet as one business object before sending it to the endpoint.	The incoming IDoc is only partially parsed (the control record of the IDoc is parsed but the data record is not). The client at the endpoint is responsible for parsing the data record.
ALE pass-through IDoc	This interface wraps the raw incoming IDoc in a business object before delivering it to the client at the endpoint. The client is responsible for parsing the raw IDoc.	On receiving the IDoc packet from SAP, the adapter wraps each raw IDoc in a business object before sending the objects, one by one, to the endpoint.	On receiving the IDoc packet from SAP, the adapter wraps the raw IDoc packet in a business object before sending it to the endpoint.	This attribute is not applicable to the ALE pass- through IDoc interface. (Neither the control record nor the data record of the IDoc is parsed.)

For more information, see the following topics.

- "Event error handling" on page 32
- "Event recovery for the ALE interface" on page 32
- "Event processing for parsed IDoc packets" on page 32
- "Event processing for unparsed IDocs" on page 34

- "IDoc status updates" on page 35

Event error handling:

WebSphere Adapter for SAP Software provides error handling for inbound ALE events by logging the errors and attempting to restart the event listener.

When the adapter detects an error condition, it performs the following actions:

1. The adapter logs the error information in the syslog (on Linux® and UNIX® systems), Windows® Event Viewer, or user trace log.
2. The adapter attempts to restart the existing event listeners.

   The adapter uses the activation specification values for RetryLimit and RetryInterval.
   - If the SAP application is not active, the adapter attempts to restart the listeners for the number of times configured in the RetryLimit property.
   - The adapter waits for the time specified in the RetryInterval parameter before attempting to restart the event listeners.
3. If the attempt to restart the event listeners fails, the adapter performs the following actions:
   - The adapter logs the error condition in the syslog (on Linux and UNIX systems), Windows Event Viewer, or user trace log.
   - The adapter cleans up the existing ALE event listeners.
   - The adapter starts new event listeners.

   The adapter uses the values of the RetryLimit and RetryInterval properties when starting the new event listeners.
4. If all the retry attempts fail, the adapter logs the relevant message and CEI events and stops trying to recover the ALE event listener.

   You must restart the adapter or Service Component Architecture (SCA) application in this case.

Event recovery for the ALE interface:

You can configure the adapter for ALE inbound processing so that it supports event recovery in case of abrupt termination.

When event recovery is specified, the adapter persists the event state in an event recovery table that resides on a data source. Event recovery is not the default behavior; you must specify it by enabling once-only delivery of events during adapter configuration.

Event processing for parsed IDoc packets:

An inbound event can contain a single IDoc or multiple IDocs, with each IDoc corresponding to a single business object. The multiple IDocs are sent by the SAP server to the adapter in the form of an IDoc packet. You can specify, during adapter configuration, whether the packet can be split into individual IDocs or whether it must be sent as one object (non-split).

Event processing begins when the SAP server sends a transaction ID to the adapter. The following sequence occurs.

1. The adapter checks the status of the event and takes one of the following actions:

- If this is a new event, the adapter stores an EVNTID (which corresponds to the transaction ID) along with a status of `0` (Created) in the event recovery table.
- If the event status is `-1` (Rollback), the adapter updates the status to `0` (Created).
- If the event status is `1` (Executed), the adapter returns an indication of success to the SAP system.

2. The SAP system sends the IDoc to the adapter.
3. The adapter converts the IDoc to a business object and sends it to the endpoint.

   For single IDocs and non-split IDoc packets, the adapter can deliver objects to endpoints that support transactions as well as to endpoints that do not support transactions.

   - For endpoints that support transactions, the adapter delivers the object as part of a unique XA transaction. When the endpoint processes the event and the transaction is committed, the status of the event is updated to `1` (Executed).

     The endpoint must be configured to support XA transactions.
   - For endpoints that do not support transactions, the adapter delivers the object to the endpoint and updates the status of the event to `1` (Executed). The adapter delivers the business object without the quality of service (QOS) that guarantees once-only delivery.
4. For split packets only, the adapter performs the following tasks:
   a. The adapter updates the BQTOTAL column (or table field) in the event recovery table to the number of IDocs in the packet. This number is used for audit and recovery purposes.
   b. The adapter sends the business objects to the message endpoint, one after the other, and updates the BQPROC property to the sequence number of the IDoc it is working on. The adapter delivers the objects to the appropriate endpoint as part of a unique XA transaction (a two-phase commit transaction) controlled by the application server.
   c. When the endpoint receives the event and the transaction is committed, the adapter increments the number in the BQPROC property.

      The message endpoint must be configured to support XA transactions.If the adapter encounters an error while processing a split IDoc packet, it can behave in one of two ways, depending on the IgnoreIDocPacketErrors configuration property:

      - If the IgnoreIDocPacketErrors property is set to `false`, the adapter stops processing any additional IDocs in the packet and reports errors to the SAP system.
      - If the IgnoreIDocPacketErrors property is set to `true`, the adapter logs an error and continues processing the rest of the IDocs in the packet. The status of the transaction is marked `3` (InProgress). In this case, the adapter log shows the IDoc numbers that failed, and you must resubmit those individual IDocs separately. You must also manually maintain these records in the event recovery table.

      This property is not used for single IDocs and for non-split IDoc packets.
   d. The SAP system sends a COMMIT call to the adapter.
   e. After the adapter delivers all the business objects in the IDoc packet to the message endpoint, it updates the event status to `1` (Executed).
   f. In case of abrupt interruptions during IDoc packet processing, the adapter resumes processing the IDocs from the current sequence number. The

adapter continues updating the BQPROC property, even if IgnoreIDocPacketErrors is set to `true`. The adapter continues the processing in case you terminate the adapter manually while the adapter is processing an IDoc packet.

5. If an exception occurs either while the adapter is processing the event or if the endpoint generates an exception, the event status is updated to `-1` (Rollback).
6. If no exception occurs, the SAP server sends a CONFIRM call to the adapter.
7. The adapter deletes the records with a 1 (Executed) status and logs a common event infrastructure (CEI) event that can be used for tracking and auditing purposes.

Event processing for unparsed IDocs:

Unparsed IDocs are passed through, with no conversion of the data (that is, the adapter does not parse the data part of the IDoc). The direct exchange of IDocs in the adapter enables high-performance, asynchronous interaction with SAP, because the parsing and serializing of the IDoc occurs outside the adapter. The consumer of the IDoc parses the IDoc.

The adapter processes the data based on whether the packet IDoc is split or non-split and whether the data needs to be parsed.

- The adapter can process packet IDocs as a packet or as individual IDocs. When an IDoc is received by the adapter from SAP as a packet IDoc, it is either split and processed as individual IDocs, or it is processed as a packet. The value of the SplitIDocPacket metadata at the business-object level determines how the IDoc is processed.

  For split IDocs, the wrapper contains only a single, unparsed IDoc object.
- The Type metadata specifies whether the data should be parsed. For unparsed IDocs, this value is `UNPARSEDIDOC`; for parsed IDocs, the value is `IDOC`. This value is set by the Adapter Connection wizard.

Unparsed data format

In the fixed-width format of an unparsed IDoc, the segment data of the IDoc is set in the IDocData field of the business object. It is a byte array of fixed-length data.

The entire segment length might not be used. The adapter pads spaces to the fields that have data; the rest of the fields are ignored, and an end of segment is set. The end of segment is denoted by a null value.

The following figure shows a segment with fields demarcated by the '|' symbol for reference.

Figure 1. Example of a segment before processing

When the adapter processes this segment into unparsed data, it takes into account only those fields that have data in them. It maintains the field width for each segment field. When it finds the last field with data, it appends a null value to mark the end of segment.

FA	FOB	VAT REG	ITA			55	null

Figure 2. Example of a segment after processing

The next segment data processed as unparsed data would be appended after the null value.

Limitations

The unparsed event feature introduces certain limitations on the enterprise application for a particular IDoc type.

- The enterprise application supports either parsed or unparsed business-object format for an IDoc type or message type.
- For an IDoc type, if you select unparsed business-object format for inbound, you cannot have inbound and outbound interfaces in the same EAR file, because outbound is based on parsed business objects.
- The DummyKey feature is not supported for unparsed IDocs.

IDoc status updates:

To monitor IDoc processing, you can configure the adapter to update the IDoc status.

When the adapter configuration property ALEUpdateStatus is set to `true` (indicating that an audit trail is required for all message types), the adapter updates the IDoc status of ALE business objects that are retrieved from the SAP server. After the event is sent to the message endpoint, the adapter updates the status of the IDoc in SAP to indicate whether the processing succeeded or failed. Monitoring of IDocs applies only to inbound processing (when the IDoc is sent from the SAP server to the adapter).

The adapter updates a status IDoc (ALEAUD) and sends it to the SAP server.

An IDoc that is not successfully sent to the endpoint is considered a failure, and the IDoc status is updated by the adapter. Similarly, an IDoc that reaches the endpoint is considered successfully processed, and the status of the IDoc is updated.

The status codes and their associated text are configurable properties of the adapter, as specified in the activation specification properties and shown in the following list:

- ALESuccessCode
- ALEFailureCode
- ALESuccessText
- ALEFailureText

Perform the following tasks to ensure that the adapter updates the standard SAP status code after it retrieves an IDoc:

- Set the AleUpdateStatus configuration property to `true` and set values for the AleSuccessCode and AleFailureCode configuration properties.

- Configure the inbound parameters of the partner profile of the logical system in SAP to receive the ALEAUD message type. Set the following properties to the specified values:

Table 1. Inbound properties of the logical system partner profile

SAP property	Value
Basic Type	ALEAUD01
Logical Message Type	ALEAUD
Function module	IDOC_INPUT_ALEAUD
Process Code	AUD1

Pass-through support for IDocs, and MQSeries link for R/3 link migration:

Both the inbound and outbound SAP adapters support a pass-through mode for IDocs.

In this mode, the bit stream for the IDoc is provided without any form of parsing. The bit stream can then be used directly in a message flow, and parsed by other parsers, or sent unchanged over transports.

Use the Adapter Connection wizard to select pass-through support: on the Configure settings for adapter pane, select ALE pass-through IDoc as the interface type.

A business object is created that contains one field, which is the bit stream of the IDoc. You can transform this business object in a Compute node to an MQSeries® link for R/3 format message, as shown in the following example.

```
DECLARE ns NAMESPACE
'http://www.ibm.com/xmlns/prod/websphere/j2ca/sap/sapmatmas05';

CREATE COMPUTE MODULE test4_Compute
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  CALL CopyMessageHeaders();
  -- CALL CopyEntireMessage();
  set OutputRoot.MQSAPH.SystemNumber = '00';
  set OutputRoot.BLOB.BLOB =
InputRoot.DataObject.ns:SapMatmas05.IDocStreamData;
  RETURN TRUE;
 END;

 CREATE PROCEDURE CopyMessageHeaders() BEGIN
  DECLARE I INTEGER 1;
  DECLARE J INTEGER;
  SET J = CARDINALITY(InputRoot.*[]);
  WHILE I < J DO
   SET OutputRoot.*[I] = InputRoot.*[I];
   SET I = I + 1;
  END WHILE;
 END;

 CREATE PROCEDURE CopyEntireMessage() BEGIN
  SET OutputRoot = InputRoot;
 END;
END MODULE;
```

You can also create a request business object from an MQSeries link for R/3 message, as shown in the following example.

```
CREATE COMPUTE MODULE test4_Compute
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  set
OutputRoot.DataObject.ns:SapMatmas05.IDocStreamData =
InputRoot.BLOB.BLOB;
  RETURN TRUE;
 END;
END MODULE;
```

The name of the SapMatmas05 element depends on selections that you make when you run the Adapter Connection wizard.

ALE pass-through IDoc business object structure:

During ALE processing, the adapter exchanges business objects with the SAP application. The Message tree contains a BLOB field that represents the IDoc.

The Message tree contains a transaction ID, a queue name, stream data, and the IDoc type. The transaction ID (SAPTransactionID) is used to ensure once-only delivery of business objects, and the queue name (qRFCQueueName) specifies the name of the queue on the SAP server to which the IDocs should be delivered. If you are not using transaction IDs or queues, these properties are blank.

ALE business objects:

A business object is a structure that consists of data, the action to be performed on the data, and additional instructions for processing the data. The adapter client uses business objects to send data to SAP or to obtain data (through the adapter) from SAP.

The adapter uses the IDoc metadata that is generated by the Adapter Connection wizard to construct a business-object definition. This metadata contains ALE-related information such as segment information, field names, and an indication of whether the business object handles a single IDoc or an IDoc packet.

The Message tree contains a BLOB field that represents the IDoc.

How business-object definitions are created

You create business-object definitions by using the Adapter Connection wizard. The wizard connects to the application, discovers data structures in the application, and generates business-object definitions to represent them. It also generates other resources that are needed by the adapter, such as the interface information that indicates the input and output parameters.

For more information, see the following topics.

- "ALE business object structure"
- "Transaction ID support" on page 38

ALE business object structure:

During ALE processing, the adapter exchanges business objects with the SAP application. The business object represents an individual IDoc or an IDoc packet. This business object is a top-level wrapper object that contains one or more IDoc child objects, each one corresponding to a single IDoc. The same business object format is used for inbound and outbound processing.

Wrapper business object

The wrapper business object contains a transaction ID, a queue name, and one or more IDoc business objects. The transaction ID (SAPTransactionID) is used to ensure once-only delivery of business objects, and the queue name (qRFCQueueName) specifies the name of the queue on the SAP server to which the IDocs should be delivered. If you are not using transaction IDs or queues, these properties are blank.

For individual IDocs, the wrapper business object contains only one instance of an IDoc business object. For IDoc packets, the wrapper business object contains multiple instances of an IDoc business object.

IDoc business object

The IDoc business object contains the following objects:

- The control record business object contains the metadata required by the adapter to process the business object.
- The data record business object contains the business object data to be processed by the SAP application and the metadata required for the adapter to convert it to an IDoc structure for the RFC call.

Unparsed IDocs

For an unparsed IDoc, in which the data part of the IDoc is not parsed by the adapter, the IDoc business object contains a dummy key, a control record, and the IDoc data.

Transaction ID support:

An SAP transaction ID (TID) is contained in the ALE wrapper business object and is therefore available as a field in the message tree. You can use transaction ID support to ensure once-only delivery of ALE objects.

The most common reason for using transaction ID support is to ensure once and only once delivery of data. The SAP transaction ID property is always generated by the Adapter Connection wizard.

The message flow must determine how to store the SAP transaction ID and how to relate the SAP transaction ID to the data being sent to the adapter. When the events are successful, the message flow should not resubmit the event associated with this TID again to prevent the processing of duplicate events.

- If the message flow does not send an SAP transaction ID with the business object, the adapter returns one after executing the transaction.
- If the message flow has an SAP transaction ID, it needs to populate the SAP transaction ID property with that value before executing the transaction.

The SAP transaction ID can be used for cross-referencing with a global unique ID that is created for an outbound event. The global unique ID is something you can create for managing integration scenarios.

Query interface for SAP Software:

The Query interface for SAP Software (QISS) provides you with the means to retrieve data from application tables on an SAP server or to query SAP application tables for the existence of data. The adapter can perform hierarchical data retrieval from the SAP application tables.

Query interface for SAP Software supports outbound interactions for read operations (RetrieveAll and Exists) only. You can use this interface in local transactions to look up records before write operations (Create, Update, or Delete). For example, you can use the interface as part of a local transaction to do an existence check on a customer before creating a sales order. You can also use the interface in non-transaction scenarios.

Query interface for SAP Software supports data retrieval from SAP application tables, including hierarchical data retrieval from multiple tables. The interface supports static as well as dynamic specification of `where` clauses for the queries.

The Adapter Connection wizard finds the application data tables in SAP, interprets the hierarchical relationship between tables, and constructs a representation of the tables and their relationship in the form of a business object. The wizard also builds a default `where` clause for the query.

You can control the depth of the data retrieval, as well as the amount of information, by using the maxRow and rowsSkip properties.

For more information, see the following topics.

- "Outbound processing for the query interface for SAP Software"
- "Business objects for the query interface for SAP Software" on page 40

Outbound processing for the query interface for SAP Software:

You use the Query interface for SAP Software for outbound processing only.

The message flow that makes the request uses the interface information that was generated by the Adapter Connection wizard.

The following list describes the sequence of processing actions that result from an outbound request that uses the query interface for SAP Software.

1. The adapter receives a request, which includes a table object, from a message flow.

   The query business object can be in a container business object, or it can be received as a table business object.
2. The adapter determines, from the table object sent with the query, the name of the table to examine.
3. The adapter determines the columns to retrieve or examine.
4. The adapter determines the rows to retrieve or examine.
5. The adapter responds.
   - For a RetreiveAll operation, the adapter returns a result set in the form of a container of query business objects, which represent the data for each row retrieved from the table. If the query is received as a table business object (not inside a container), the rows are returned one at a time, as they are retrieved.
   - For the Exists operation, the adapter returns an indication of whether the data exists in the SAP table.

- If no data exists, the adapter generates an exception.

Business objects for the query interface for SAP Software:

A business object is a structure that consists of data, the action to be performed on the data, and additional instructions, if any, for processing the data. The input to the Query interface for SAP Software is a table business object. The table business object represents the columns in a table on the SAP server. The adapter uses the table business object to obtain data from tables on the SAP server.

How data is represented in business objects

The adapter uses metadata that is generated by the Adapter Connection wizard to construct a business-object definition.

The data in the business object represents the columns of the associated table in SAP.

How business objects are created

You create business-object definitions by using the Adapter Connection wizard. The wizard connects to the application, discovers data structures in the application, and generates business-object definitions to represent them. It also generates other resources that are needed by the adapter, such as the interface information that indicates the input and output parameters.

Business object structure

The table business object can be part of a container.

The table business object contains columns selected from the specified SAP table.

In addition to column information, the table business object also contains a query business object as the last parameter.

The properties of the query business object are sapWhereClause, sapRowsSkip, and sapMaxRows:

- The sapWhereClause property retrieves information from SAP tables. The default value is populated by the Adapter Connection wizard. The space character is used as the delimiter to parse the sapWhereClause.
- The sapMaxRows property is the maximum number of rows to be returned. The default value is `100`.
- The sapRowsSkip property is the number of rows to skip before retrieving data. The default value is `0`.

The tables can be modeled as hierarchical business objects. You specify the parent-child relationship of the tables in the Adapter Connection wizard.

Tables are linked by a foreign key to form parent-child relationships. The child table business object has a foreign key that references a property in the parent query business object.

In the KNA1 business object, notice the reference to SapAdrc, a child business object. The SapAdrc table object, shown in the following figure, has a column

named AddressNumber. This column has an associated property (ForeignKey) that contains a reference to the parent business object.

The return from the Query interface for SAP Software call for a RetrieveAll operation is a container of table objects.

The Advanced event processing interface:

The Advanced event processing interface of the WebSphere Adapter for SAP Software is used for both inbound and outbound processing.

For inbound processing, it polls for events in SAP, converts them into business objects, and sends the event data as business objects to WebSphere Message Broker.

For outbound processing, the adapter processes events sent from an application to retrieve data from or update data in the SAP server.

For more information, see the following topics.

- "Outbound processing for the AEP interface"
- "Inbound processing for the AEP interface" on page 44
- "Business objects for the AEP interface" on page 47

Outbound processing for the AEP interface:

During outbound processing, business object data is converted into an ABAP handler function, which is called on the SAP server. When the data is returned by the ABAP handler function, the data is converted to a business object, and the business object is returned as a response.

The following list describes the sequence of processing actions that result from an outbound request that uses the Advanced event processing interface.

1. The adapter receives the Advanced event processing record, which contains business data along with the metadata.
2. The Advanced event processing interface of the adapter uses the metadata of the business object to obtain the type of IDoc specified and to reformat the business object data into the structure of that IDoc.
3. After it reformats the data, the adapter passes the business object data to an object-specific ABAP handler (based on the operation), which handles the integration with an SAP native API.
4. After the object-specific ABAP handler finishes processing the business object data, it returns the response data in IDoc format to the adapter, which converts it to the business object.
5. The adapter returns the results to the caller.

For more information, see the following topics.

- "ABAP handler overview"
- "ABAP handler creation" on page 43
- "Call Transaction Recorder wizard" on page 44

ABAP handler overview:

An ABAP handler is a function module that gets data into and out of the SAP application database. For each business object definition that you develop, you must support it by developing a custom ABAP handler.

ABAP handlers are in the SAP application as ABAP function modules. ABAP handlers are responsible for adding business-object data into the SAP application database (for Create, Update, and Delete operations) or for using the business-object data as the keys to retrieving data from the SAP application database (for the Retrieve operation).

You must develop operation-specific ABAP handlers for each hierarchical business object that must be supported. If you change the business object definition, you must also change the ABAP handler.

An ABAP handler can use any of the SAP native APIs for handling the data. The following list contains some of the native APIs.

- Call Transaction

  Call Transaction is the SAP-provided mechanism for entering data into an SAP system. Call Transaction guarantees that the data adheres to the SAP data model by using the same screens an online user sees in a transaction. This process is commonly referred to as *screen scraping*.
- Batch data communication (BDC)

  Batch Data Communication (BDC) is an instruction set that SAP can follow to process a transaction without user intervention. The instructions specify the sequence in which the screens in a transaction are processed and which fields are populated with data on which screens. All of the elements of an SAP transaction that are exposed to an online user have identifications that can be used in a BDC.
- ABAP SQL

  ABAP SQL is the SAP proprietary version of SQL. It is database-independent and platform-independent, so that whatever SQL code you write, you can run it on any database and platform combination that SAP supports. ABAP SQL is similar in syntax to other versions of SQL and supports all of the basic database table commands such as update, insert, modify, select, and delete. For a complete description of ABAP SQL, see your SAP documentation.

  By using ABAP SQL, an ABAP handler can modify SAP database tables with business-object data for create, update, and delete operations. It can also use the business-object data in the where clause of an ABAP select statement as the keys.

  Do not use ABAP SQL to modify SAP tables, because it might corrupt the integrity of the database. Use ABAP SQL only to retrieve data.
- ABAP Function Modules and Subroutines

  From the ABAP handler, you can call ABAP function modules or subroutines that implement the required function.

The adapter provides the following tools to help in the development process:

- The adapter includes the Call Transaction Recorder wizard to assist you in developing the ABAP handlers that use call transactions or BDC sessions.
- The Adapter Connection wizard generates the required business objects and other resources for Advanced event processing. The business objects are based on IDocs, which can be custom or standard.
- The adapter provides samples that you can refer to for an understanding of the Advanced event processing implementation.

ABAP handler creation:

For each IDoc object definition that you develop, you must support it by developing a custom ABAP handler.

You can use either standard IDocs or custom IDocs for the Advanced event processing interface. After defining the custom IDoc for an integration scenario, create an ABAP handler (function module) for each operation of the business object that must be supported.

Each function should have the following interface to ensure that adapter can call it:

```
*" IMPORTING
*" VALUE(OBJECT_KEY_IN) LIKE /CWLD/LOG_HEADER-OBJ_KEY OPTIONAL
*" VALUE(INPUT_METHOD) LIKE BDWFAP_PAR-INPUTMETHD OPTIONAL
*" VALUE(LOG_NUMBER) LIKE /CWLD/LOG_HEADER-LOG_NR OPTIONAL
*" EXPORTING
*" VALUE(OBJECT_KEY_OUT) LIKE /CWLD/LOG_HEADER-OBJ_KEY
*" VALUE(RETURN_CODE) LIKE /CWLD/RFCRC_STRU-RFCRC
*" VALUE(RETURN_TEXT) LIKE /CWLD/LOG_HEADER-OBJ_KEY
*" TABLES
*" IDOC_DATA STRUCTURE EDID4
*" LOG_INFO STRUCTURE /CWLD/EVENT_INFO
```

The following table provides information about the parameters:

Table 2. Interface parameters

Parameter	Description
OBJECT_KEY_IN	Should be no value.
INPUT_METHOD	Indicates whether the IDoc should be processed in a dialog (that is, through Call Transaction). Possible values are: " " - Background (no dialog) "A" - Show all screens "E" - Start the dialog on the screen where the error occurred “N” Default
LOG_NUMBER	Log Number.
OBJECT_KEY_OUT	Customer ID returned from the calling transaction.
RETURN_CODE	0 - Successful. 1 - Failed to retrieve. 2 - Failed to create, update, or delete.
RETURN_TEXT	Message describing the return code.
IDOC_DATA	Table containing one entry for each IDoc data segment. The following fields are relevant to the inbound function module: Docnum - The IDoc number. Segnam - The segment name. Sdata - The segment data.
LOG_INFO	Table containing details regarding events processed with either a success or error message.

Call Transaction Recorder wizard:

The adapter provides the Call Transaction Recorder wizard to assist you in developing the ABAP handlers that use call transactions or BDC sessions.

The Call Transaction Recorder wizard enables you to generate sample code for call transactions to facilitate development. It generates sample code stubs for each screen that is modified during the recording phase.

To access this wizard, enter the /CWLD/HOME transaction in the SAP GUI.

The following example is sample code that is generated by the wizard. You can adopt this code in the ABAP Handler.

```
* Customer master: request screen chnge/displ cent.
perform dynpro_new using 'SAPMF02D' '0101' .

* Customer account number
perform dynpro_set using 'RF02D-KUNNR' '1' .

* Function Command
perform dynpro_set using 'BDC_OKCODE' '/00' .

* Function Command
perform dynpro_set using 'BDC_OKCODE' '/00' .

* Customer master: General data, CAM address, communication
perform dynpro_new using 'SAPMF02D' '0111' .

* Title
perform dynpro_set using 'SZA1_D0100-TITLE_MEDI' 'Mr.' .

* Function Command
perform dynpro_set using 'BDC_OKCODE' '=UPDA' .

* Call Transaction
Call Transaction 'XD02' using bdcdata
    mode input_mode
   update 'S'
  messages into bdc_messages.
```

The wizard does not generate the required business object. You use the Adapter Connection wizard to generate the business object.

Inbound processing for the AEP interface:

The adapter uses the Advanced event processing interface to poll for events on the SAP server, to process the events, and to send them to an endpoint.

The following list describes the sequence of processing actions that result from an inbound request that uses the Advanced event processing interface.

1. A triggered event enters the event table with an initial status of prequeued.
2. When the adapter polls for events, the status of the event changes from prequeued to queued if there are no database locks for the combination of the user who created the event and the event key.
3. After the event is retrieved from the event table, the status of the event is updated to InProgress.

If locks exist, the status of the event is set to locked and the event is requeued into the queue. Every event with a prequeued or locked status is updated with every poll. You can configure the polling frequency by using the Poll Frequency property.

4. After preprocessing all prequeued events, the adapter selects the events.

   The property Poll Quantity determines the maximum number of events returned for a single poll call.

5. For each event, the adapter uses the remote function specified for the Retrieve operation to retrieve the data and send it to the endpoint.

   If the AssuredOnceDelivery property is set to true, an XID value is set for each event in the event store. After each event is picked up for processing, the XID value for that event is updated in the event table.

   If, before the event is delivered to the endpoint, the SAP connection is lost or the application is stopped, and the event is consequently not processed completely, the XID column ensures that the event is reprocessed and sent to the endpoint. After the SAP connection is reestablished or the adapter starts up again, it first checks for events in the event table that have a value in the XID column. It then processes these events first and then polls the other events during the poll cycles.

6. After each event is processed, it is updated or archived in the SAP application.

   When the event is processed successfully, it is archived and then deleted from the event table.

   The adapter can also filter the events to be processed by business object type. The filter is set in the Event Filter Type property. This property has a comma-delimited list of business object types, and only the types specified in the property are picked for processing. If no value is specified for the property, no filter is applied and all the events are picked up for processing.

For more information, see the following topics.

- "Event detection"
- "Event triggers" on page 47

Event detection:

Event detection refers to the collection of processes that notify the adapter of SAP application object events. Notification includes, but is not limited to, the type of the event (object and operation) and the data key required for the external system to retrieve the associated data.

Event detection is the process of identifying that an event was generated in the SAP application. Typically, adapters use database triggers to detect an event. However, because the SAP application is tightly integrated with the SAP database, SAP allows very limited access for direct modifications to its database. Therefore, the event-detection mechanisms are implemented in the application transaction layer above the database.

Adapter-supported event detection mechanisms

The four event-detection mechanisms that are supported by the adapter are described in the following list:

- Custom Triggers, which are implemented for a business process (normally a single SAP transaction) by inserting event detection code at an appropriate point in the SAP transaction

- Batch programs, which involve developing an ABAP program containing the criteria for detecting an event
- Business workflows, which use the object-oriented event detection capabilities of SAP
- Change pointers, a variation of business workflows, which use the concept of change documents to detect changes for a business process

All these event-detection mechanisms support real-time triggering and retrieval of objects. In addition, custom triggers and batch programs provide the ability to delay the retrieval of events. An event whose retrieval is delayed is called a future event.

Each event detection mechanism has advantages and disadvantages that need to be considered when designing and developing a business object trigger. Keep in mind that these are only a few examples of event detection mechanisms. There are many different ways to detect events.

After you determine the business process to support (for example, sales quotes or sales orders) and determine the preferred event-detection mechanism, implement the mechanism for your business process.

When implementing an event detection mechanism, it is a good idea to support all of the functions for a business process in one mechanism. This limits the effect in the SAP application and makes event detection easier to manage.

Event table

Events that are detected are stored in an SAP application table. This event table is delivered as part of the ABAP component. The event table structure is as follows.

Table 3. Event table fields

Name	Type	Description
event_id	NUMBER	Unique event ID that is a primary key for the table.
object_name	STRING	Business graph name or business object name.
object_key	STRING	Delimited string that contains the keys for the business object.
object_function	STRING	Operation corresponding to the event (Delete, Create, or Update).
event_priority	NUMBER	Any positive integer to denote the priority of the event.
event_time	DATE	Date and time when the event was generated.
event_status	NUMBER	Event processing status. Possible values are: 0 - Ready for poll 1 - Event delivered 2 - Event prequeued 3 - Event in progress 4 - Event locked -1 - Event failed
Xid	STRING	Unique XID (transaction ID) value for assured-once delivery.
event_user	STRING	User who created the event.

Table 3. Event table fields (continued)

Name	Type	Description
event_comment	STRING	Description of the event.

Event triggers:

After an event is identified by one of the event-detection mechanisms, it is triggered by one of the adapter-delivered event triggers. Event triggers can cause events to be processed immediately or in the future.

The function modules that trigger events are described in the following list.

- /CWLD/ADD_TO_QUEUE

  This function module triggers events to the current event table for immediate processing.
- /CWLD/ADD_TO_QUEUE_IN_FUTURE

  This function module triggers events to the future event table to be processed at a later time.

Both functions are for real-time triggering.

Current event table

If the event will be triggered in real-time, /CWLD/ADD_TO_QUEUE_AEP commits the event to the current event table (/CWLD/EVT_CUR_AEP). Specifically, it adds a row of data for the object name, verb, and key that represents the event.

Future event table

If an event needs to be processed at a future date, the processing that is described in the following list occurs.

1. A custom ABAP handler calls /CWLD/ADD_TO_QUEUE_IN_FUTURE_AEP with the event.
2. The /CWLD/ADD_TO_QUEUE_IN_FUTURE_AEP module commits the event to the future event table (/CWLD/EVT_FUT_AEP). Specifically, it adds a row of data for the object name, verb, and key that represents the event. In addition, it adds a Date row
3. The adapter-delivered batch program /CWLD/SUBMIT_FUTURE_EVENTS_AEP reads the future event table.
4. If scheduled to do so, the batch program retrieves events from the future event table.
5. After it retrieves an event, the batch program calls /CWLD/ADD_TO_QUEUE_AEP.
6. The /CWLD/ADD_TO_QUEUE_AEP module triggers the event to the current event table.

/CWLD/ADD_TO_QUEUE_IN_FUTURE_AEP uses the system date as the current date when it populates the Date row of the future event table.

Business objects for the AEP interface:

A business object is a structure that consists of data, the action to be performed on the data, and additional instructions, if any, for processing the data.

How data is represented in business objects

Advanced event processing business objects are based on custom IDocs, standard IDocs, or extension IDocs available in the SAP system.

How business-object definitions are created

You create business-object definitions by using the Adapter Connection wizard. The wizard connects to the application, discovers data structures in the application, and generates business-object definitions to represent them. It also generates other resources that are needed by the adapter, such as the interface information that indicates the input and output parameters.

For custom interfaces that you want to support, as a first step, you must define the custom IDoc in the SAP system. You can then use the Adapter Connection wizard to discover this custom IDoc and build the required resources, including the business-object definition.

Overview of WebSphere Adapter for Siebel Business Applications

With WebSphere Adapter for Siebel Business Applications, you can create integrated processes that exchange information with a Siebel application, without special coding.

Use the WebSphere Adapter for Siebel Business Applications to create integrated processes that exchange information with a Siebel application. With the adapter, an application can send requests to the Siebel Business Applications server or receive notifications of changes from the server.

The adapter creates a standard interface to the applications and data on the Siebel Business Applications server, so that the application developer does not need to understand the lower-level details (the implementation of the application or the data structures) on the Siebel Business Applications server. An application, for example, can send a request to the Siebel Business Applications server, to query or update an Account record, represented by a Siebel business component instance. It can also receive events from the server; for example, to be notified that a customer record has been updated. This behavior provides you with improved business workflow and processes to help manage your customer relations.

WebSphere Adapter for Siebel Business Applications complies with the Java Connector Architecture (JCA). JCA standardizes the way application components, application servers, and Siebel applications, such as Siebel Business Applications server, interact with each other.

The adapter configuration, which you generate with the Adapter Connection wizard, uses a standard interface and standard data objects. The adapter takes the standard data object sent by the application component and calls the Siebel Business Applications function. The adapter then returns a standard data object to the application component. The application component does not have to deal directly with the Siebel Business Applications function; it is the Siebel Business Applications adapter that calls the function and returns the results.

For example, the application component that needs the list of customers sends a standard business object with the range of customer IDs to Adapter for Siebel Business Applications. In return, the application component receives the results (the list of customers) in the form of a standard business object. The application component does not need to know how the function worked or how the data was structured. The adapter completes all the interactions with the Siebel Business Applications function.

Similarly, the message flow might want to know about a change to the data on the Siebel Business Applications server (for example, a change to a particular customer). You can generate an adapter component that polls for such events on the Siebel Business Applications server and notifies message flows of the update. In this case, the interaction begins at the Siebel Business Applications server.

Technical overview of the Adapter for Siebel Business Applications:

WebSphere Adapter for Siebel Business Applications supports the exchange of information between your existing applications and Siebel Business Applications. The adapter supports Siebel entities, including business objects, business components, and business services. This enables you to create business processes that exchange data.

The adapter supports outbound processing (requests for data or services from an application to the Siebel application) and inbound processing (event notification from a Siebel application server to an application).

With Adapter for Siebel Business Applications, you can use existing or newly-created applications that run in a supported runtime environment to send requests for data and services to Siebel Business Applications.

You can also add event-generation triggers to Siebel business objects to have notifications of events, such as the creation, update, and deletion of a record, sent to one or more of your applications.

For more information, see the following topics.

- "Outbound processing for Siebel Business Applications"
- "Inbound processing for Siebel Business Applications" on page 50
- "Business objects for Siebel Business Applications" on page 53
- "Adapter Connection wizard (Siebel)" on page 54

Outbound processing for Siebel Business Applications:

WebSphere Adapter for Siebel Business Applications supports synchronous outbound processing. This means that when the component sends a request in the form of a WebSphere business object hierarchy to the adapter, the adapter processes the request and returns a WebSphere business object hierarchy that represents the result of the operation.

When the adapter receives a WebSphere business object hierarchy, the adapter processes it as follows:

1. The adapter extracts metadata from the WebSphere business object hierarchy.
2. It identifies the appropriate Siebel objects to access (for example, Siebel business objects and business components, or Siebel business services, integrations objects, and integration components) depending on the objects against which the artifacts were generated.

3. The adapter extracts the outbound operation to perform from the WebSphere business object hierarchy.
4. After accessing the required Siebel objects, the adapter retrieves, updates, deletes, or creates a Siebel business component hierarchy or performs the corresponding business service method on the integration component hierarchy.
5. If there are updates (Create, Update, Delete), the adapter populates that Siebel object (business or integration component hierarchy) with data from the hierarchy of WebSphere business objects.

Supported Outbound Operations

WebSphere Adapter for Siebel Business Applications supports the outbound operations shown in the following table.

Table 4. Supported outbound operations

Operation	Description
Create	Creates the business component.
Delete	Deletes the business component and its children.
Exists	Checks for the existence of incoming business objects. The output business object, "ExistsResult" will be returned with the boolean value populated.
Retrieve	Specifies the value of the business component.
RetrieveAll	Retrieves multiple instances of the same business component and populates it as the container business graph.
Update	Updates the Siebel application with the incoming business object.

Inbound processing for Siebel Business Applications:

WebSphere Adapter for Siebel Business Applications supports asynchronous inbound processing, which means that the adapter polls the Siebel Business Applications at specified intervals for events. When the adapter detects an event, it converts the event data into a business object and sends it to the component.

Before inbound processing can occur, a Siebel event business component must be created in the Siebel application (IBM2 for Siebel version 7.x and IBM_EVENT for Siebel version 8) and its name specified against the corresponding property in the adapter activation specification.

When the adapter detects an event for Siebel event business components or integration components, it processes the event by retrieving the updated data for the Siebel event business component or integration component and converting it into a business object. The adapter then sends the business object to the event business component. For example, if an event business component (an account) is updated, an event trigger adds an event record to the event business component. The adapter polls the event business component, retrieves the event record, and processes it.

When the adapter finds an event for a Siebel event business component, it processes the event in the following way:

1. The adapter retrieves the event information from the Siebel event business component.

2. The adapter retrieves the corresponding event business component instance hierarchy.
3. The adapter populates the associated WebSphere business object or business graph (if it was generated) with the values that it retrieves from the event business component.
4. The adapter sends a notification to each registered application.

If an inbound event in the event table fails or is invalid, the event status is updated to -1, which indicates an error in processing the event, and a resource exception message is issued that explains the reason for the error.

Event store for Siebel Business Applications:

The event store is a persistent cache where event records are saved until the polling adapter can process them. To keep track of inbound events as they make their way through the system, the adapter uses an event store.

The creation, update, or deletion of an event record in the Siebel business application is an 'event'. Each time a business object is created, updated, or deleted, the adapter updates the status of the event in an event store.

For example, if you have a customer component and a new customer has just been added to it, this signals an update. If the adapter is configured to receive the events about the new update, there are triggers attached to the Siebel end and connected to the customer component. The triggers add a record to the event business component. The record contains information about the new customer, such as the customer ID. This information is stored in the object key. The object key is the unique identifier that provides the key name and value of the event business component that was updated (for example, `Id=1-20RT`). The object name is the WebSphere business-object name that represents the customer component (for example, `AccountBG` or `Account`). The adapter retrieves this event and the new customer information that relates to it. It then processes the event and delivers it to the export component.

During inbound processing, the adapter polls the event business components from the event store at regular intervals. Each time it polls, a number of events are processed by the adapter. Events are processed in ascending order of priority and ascending order of the event time stamp. In each poll cycle, new events are picked up. The adapter retrieves the value set in the object key field for the event and loads the business object that corresponds to it. The business object, or optionally the business graph, is created from the retrieved information and is delivered to the export components.

If you set the activation specification property AssuredOnceDelivery to `true`, a transaction ID (XID) value is set for each event in the event store. After the event is retrieved for processing, the XID value for it is updated in the event store and displayed in the XID column in the event business component. The event is then delivered to its corresponding export component, and the status is updated to show that the event has been successfully delivered. If the application is stopped or the event is not processed completely, the XID column is filled with a value. This ensures that the event is reprocessed and sent to the export component. After the connection is reestablished or the adapter starts again, the adapter checks for events in the event store that have a value in the XID column. The adapter processes these events first, then polls the other events during the poll cycles.

The adapter can either process all events or process events filtered by business-object type. You set the filter through the activation specification property, EventTypeFilter. This property contains a comma-delimited list of business-object types. Only the types specified in the property are processed. If the EventTypeFilter property is not set, all of the events are processed. If the FilterFutureEvents property is set to `true`, the adapter filters events based on the time stamp. The adapter compares the system time in each poll cycle to the time stamp on each event. If an event is set to occur in the future, it is not processed until that time.

After an event is successfully posted and delivered to the export component, the entry is deleted from the event store. Failed events (posting and delivery to the export component is unsuccessful), remain in the event store and are marked `-1`. This prevents duplicate processing.

Event store structure for Siebel business objects and business components

The `IBM2` event business component stores information about the event. The information stored is used by the resource adapter during event subscription to build the corresponding business object and send it to the registered export components. The information that is stored, and the structure of the event store used by the adapter, are shown in the following table.

Table 5. Event store structure for `IBM2` Siebel event business objects and business components

Field	Description	Example
Description	Any comment associated with the event.	Account Create Event
Event ID	The ID of the event row.	Automatically generated unique ID in Siebel (for example: `1-XYZ`)
Event timestamp	The time stamp for the event. The format is in *mm/dd/yyyy hh:mm:ss*	*02/24/2007 11:37:56*
Event type	The type of event.	Create, Update, or Delete
Object key	A unique identifier of the business-object row for which the event was created. It is a name-value pair consisting of the name of the property (key name) and value.	Id=1-20RT
Object name	The name of the business object or business graph for which the event was detected.	`IOAccountPRMANIICAccount`
Priority	The event priority.	1
Status	The event status. This is initially set to the value for a new event and updated by the adapter as it processes the event. The status can have one of the following values: • 0: Identifies a new event. • 1: Identifies an event that has been delivered to an export component. • -1: An error occurred while processing the event. This column cannot contain a null value.	0

Table 5. Event store structure for IBM2 Siebel event business objects and business components (continued)

Field	Description	Example
XID	The transaction ID. This is to ensure assured once-only delivery.	None

Event store structure for Siebel business services

The event is retrieved from the IBM2 event business component and the information is used to retrieve the event business component. This creates a business graph which is published to the registered export components.

Table 6. Event store structure for IBM2 Siebel business services

Field	Description	Example
Description	Any comment associated with the event.	Account PRM ANI Event
Event ID	The ID of the event row.	Automatically generated unique ID in Siebel (for example: 1-XYZ)
Event timestamp	The time stamp for the event. The format is in *mm/dd/yyyy hh:mm:ss*	*02/24/2007 11:37:56*
Event type	The type of event.	Create, Update, or Delete
Object key	A unique identifier of the business-object row for which the event was created. It is a name value pair consisting of the name of the property (key name) and value.	Name=TestName;Location=BGM, where 'Name' and 'Location' are the keys in the integration component. 'TestName' and 'BGM' are the values specified, and ; is the event key delimiter.
Object name	The name of the business object or business graph for which the event was detected.	IOAccountPRMANIICAccount
Priority	The event priority.	1
Status	The event status. This is initially set to the value for a new event and updated by the adapter as it processes the event. The status can have one of the following values: • 0: Identifies a new event. • 1: Identifies an event that has been delivered to an export component. • -1: An error occurred while processing the event. This column cannot contain a null value.	0
XID	The transaction ID. This is to ensure 'assured once delivery'.	None

Business objects for Siebel Business Applications:

To send data or obtain data from Siebel Business Applications, the adapter uses business objects. A business object is a structure that consists of data, the action to

be performed on the data, and additional instructions, if any, for processing the data. The data can represent either a business entity, such as an invoice or an employee record, or unstructured text.

How business objects are created

You create business objects by using the Adapter Connection wizard, which connects to the application, discovers data structures in the application, and generates business objects to represent them. It also generates other resources that are needed by the adapter.

The Siebel business objects are created with long names by default. To generate business objects with shorter names, select **Generate business objects with shorter names** on the Configure Objects screen of the Adapter Connection wizard. For more information, see "Naming conventions for business objects representing Siebel business services" on page 1404.

Business object structure

The adapter supports business objects that are hierarchically structured. The top-level business object must have a one-to-one correspondence with the Siebel business component, and collections that occur in the top-level object are children of it. Information about the processed object is stored in the application-specific information for the object and each of its attributes.

Adapter Connection wizard (Siebel):

The Adapter Connection wizard is a tool that you use to configure your adapter. The wizard establishes a connection to the Siebel server, discovers business objects and services (based on search criteria you provide), and generates business objects based on the services discovered.

By using the Adapter Connection wizard, you establish a connection to the Siebel server to browse the metadata repository on the Siebel server. You also specify connection information, such as the Connection URL, user name, and password needed to access the server.

The result of running the wizard is a message set project that contains the Siebel business objects and services along with the adapter.

Overview of WebSphere Adapter for PeopleSoft Enterprise

With the adapter for PeopleSoft Enterprise you can create integrated processes that exchange information with PeopleSoft Enterprise through a standard interface. This interface shields the message flow developer from having to understand the lower level details regarding implementation or the application or data structures it uses.

With the adapter, a message flow can send a request, for example to a PeopleSoft Enterprise database to query a record in an HR table, or it can receive events from the server, such as notification that an employee record has been updated.

WebSphere Adapter for PeopleSoft Enterprise complies with the Java Connector Architecture (JCA), which standardizes the way application components, application servers, and enterprise information systems, such as a PeopleSoft Enterprise server, interact with each other.

The adapter component, which you generate with the Adapter Connection wizard uses a standard interface and standard data objects. The adapter component takes the standard data object sent by the message flow and calls the PeopleSoft function. It then returns a standard data object to the message flow. The message flow does not have to deal directly with the PeopleSoft function; it is the adapter component that calls the function and returns the results.

For example, the message flow that requested the list of employees sends a standard business object with the range of skill codes to the PeopleSoft adapter component. The message flow receives, in return, the results (the list of employees) in the form of a standard business object. The adapter component completes all the interactions directly with the PeopleSoft function.

Similarly, the message flow might need to be notified about a change to the data on the PeopleSoft Enterprise server (for example, a change to the skills set of a particular employee). You can generate an adapter component that listens for such events on the PeopleSoft Enterprise server and notifies message flows with the update. In this case, the interaction begins at the PeopleSoft Enterprise server.

For more information, see "Technical overview of the WebSphere Adapter for PeopleSoft Enterprise."

Technical overview of the WebSphere Adapter for PeopleSoft Enterprise:

The adapter supports the exchange of business data between the PeopleSoft Enterprise server and WebSphere Message Broker. It does so by connecting to two layers of PeopleTools application programming interface classes that reveal the underlying business data for integration.

Adapter for PeopleSoft Enterprise establishes bidirectional connectivity with the PeopleSoft Enterprise server by connecting to two PeopleTools application programming interfaces as follows:

1. The adapter accesses the primary API layer to create a session instance and to connect to the application server through the Jolt port.
2. The adapter then accesses the PeopleSoft Component Interface API, which reveals underlying business objects, logic, and functions.

In PeopleSoft, a component is a set of pages grouped together for a business purpose (such as an employee profile), and a component interface is an API that provides synchronous access to a component from an external application. After the adapter connects to the component interface, the following entities are exposed to the adapter and available for integration:

- All business objects in the component interface definition
- PeopleCode methods associated with the underlying components
- Records, except searches and menu-specific processing options

For more information, see the following topics.

- "Outbound processing for PeopleSoft Enterprise"
- "Inbound processing for PeopleSoft Enterprise" on page 56
- "Business objects for PeopleSoft Enterprise" on page 58

Outbound processing for PeopleSoft Enterprise:

The Adapter for PeopleSoft Enterprise supports synchronous outbound request processing. Synchronous outbound processing means that when the message flow sends a request in the form of a business object to the adapter, the adapter processes the request and returns a business object representing the result of the operation to the message flow.

When the adapter receives a WebSphere business object hierarchy, adapter processes it as follows:

1. The adapter extracts metadata from the WebSphere business object hierarchy that identifies the appropriate PeopleSoft component interface to access.
2. The adapter extracts the outbound operation to perform from the WebSphere business object hierarchy.
3. Once it accesses the component interface, the adapter sets the keys from values specified in the business objects. If key values are not generated, for example with a create operation, the PeopleSoft application generates key fields.
4. After it retrieves the PeopleSoft objects, the adapter instantiates an existing component interface to delete, retrieve, update, or create a component interface.
5. If there are updates (Create, Update), the adapter populates the component interface with data from the WebSphere business object hierarchy. If there are Deletes, the adapter populates the component interface only with StatusColumnName and value information.

The adapter processes attributes in the order defined in the business object. For example, if there is a complex attribute between two simple attributes, the adapter processes the simple attribute at the first position, the complex attribute followed by the simple attribute. After the changes are made, the component interface is saved to commit the data to the PeopleSoft database. This pattern of processing is used for Create and Update operations only.

Supported outbound operations

WebSphere Adapter for PeopleSoft Enterprise supports the following outbound operations:

Table 7. Supported outbound operations

Operation	Description
Create	Creates the business object.
Delete	Deletes the business object and its children. Because the adapter supports only logical deletes, objects are marked as deleted but not removed.
Exists	Checks for the existence of incoming business objects.
Retrieve	Retrieves the PeopleSoft component, and maps component data onto the business object hierarchy.
RetrieveAll	Retrieves multiple instances of the PeopleSoft component, and maps component data onto the business object hierarchy.
Update	Updates the corresponding PeopleSoft component with the incoming business object.

Inbound processing for PeopleSoft Enterprise:

The WebSphere Adapter for PeopleSoft Enterprise supports inbound event processing.

Inbound event processing means that the adapter polls the PeopleSoft Enterprise server at specified intervals for events. When the adapter detects an event, it converts the event data into a business object and sends it to the message flow.

To use inbound event processing, you must create a custom event project in PeopleSoft, as described in "Creating a custom event project in PeopleTools" on page 296.

For more information, see "Event store for PeopleSoft Enterprise."

Event store for PeopleSoft Enterprise:

The event store is a table that holds events that represent data changes until the polling adapter can process them. The adapter uses the event store to keep track of event entities.

To use inbound processing, you must use PeopleTools Application Designer to create a custom project for event notification. The custom project uses two PeopleCode functions that determine the way future events are processed, and the custom project creates the event store the adapter needs for inbound processing. Each time a business object is created, updated, or deleted, the PeopleCode function used in the project and then added to the component interface inserts a new record in the event store, with the appropriate object name, keys, and status value.

With inbound processing, the adapter polls the event entities from the event store at configured poll intervals. In each poll call, a configured number of events are processed by the adapter. The order of event processing is based on the ascending order of priority and the ascending order of the event time stamp. The events with the status, Ready for poll (0), are picked up for polling in each poll cycle. The adapter uses the object name and object key to retrieve the corresponding business object.

If you set the activation specification property AssuredOnceDelivery to `true`, an XID (transaction ID) value is set for each event in the event store, and it is used to ensure that an event is delivered only once to the target application. After an event is obtained for processing, the XID value for that event is updated in the event store. The event is then delivered to its corresponding export component, and its status is updated to show that event delivery has been completed. If the application is stopped before the event can be delivered to the export component or if delivery has failed, the event might not be processed completely. In this case, the XID value represents in-progress status, and the XID column ensures that the event is reprocessed and sent to the export component. Once the database connection is re-established or the adapter starts again, the adapter checks for events in the event table that have a value in the XID column of Ready for Poll (0). The adapter processes these events first, and then polls the other events during the poll cycles.

The adapter uses special processing for events that have status code (99), which indicates that they will occur in the future. During a poll cycle, when the adapter retrieves events with a future status, the adapter compares the system time with the time stamp on each event. If the event time is earlier than or equal to the system time, the adapter processes the event and changes the event status to Ready for Poll (0).

If you want to the adapter to process future status events in the present time, use the function IBM_PUBLISH_EVENT instead of IBM_FUTURE_PUBLISH_EVENT. Doing so means that the event is identified as Ready to Poll (0) instead of Future (99).

As events are retrieved and processed from the event store, the status of the event changes to reflect the cycle, as shown in the following table:

Table 8. Event status values

Status short name	Description	Event table value
Error processing event	An error occurred during event processing.	-1
Ready for poll	The event has not yet been picked up by the adapter. The event is ready to be picked up.	0
Success	The event has been delivered to the event manager.	1
Deleted	The event has been processed successfully and should be removed from the event store.	4
Future Events	These events should be processed at a future date.	99

Business objects for PeopleSoft Enterprise:

To send data or obtain data from PeopleSoft Enterprise, the adapter uses business objects. A business object is a structure that consists of data, the action to be performed on the data, and additional instructions, if any, for processing the data. The data can represent either a business entity, such as an invoice or an employee record, or unstructured text.

How business objects are created

You create business objects by using the Adapter Connection wizard. The wizard connects to the application, discovers data structures in the application, and generates business objects to represent them. It also generates other resources that are needed by the adapter.

Business object structure

The adapter supports business objects that are hierarchically structured. The top-level business object must have a one-to-one correspondence with the PeopleSoft component interface, and collections that occur in the top-level object are children of it. Information about the processed object is stored in the application-specific information for the object and each of its attributes.

The following table describes the attributes that form a business object.

Attribute property	Description
Name	Indicates the name of the Business Object attribute.

Attribute property	Description
Type	Indicates the type of the Business Object attribute. The adapter uses character mapping between PeopleSoft component property types and the generated business object attribute types. PeopleSoft component property types map to generated attribute types in the following manner: CHAR maps to attribute type String NUMBER maps to attribute type BigDecimal LONG maps to attribute type Long SIGN maps to attribute type BigDecimal DATE maps to attribute type Date TIME maps to attribute type Time DTTM maps to attribute type DateTime
Key	Child business objects have their own keys that have the primary key application-specific information. They also inherit keys from their parent business object.
Cardinality	Single cardinality for simple attributes; multiple cardinality for container attributes.

IBM Information Management System (IMS)

IMS™ is a message-based transaction manager and hierarchical-database manager for z/OS®. External applications can use transactions to interact with applications that run inside IMS.

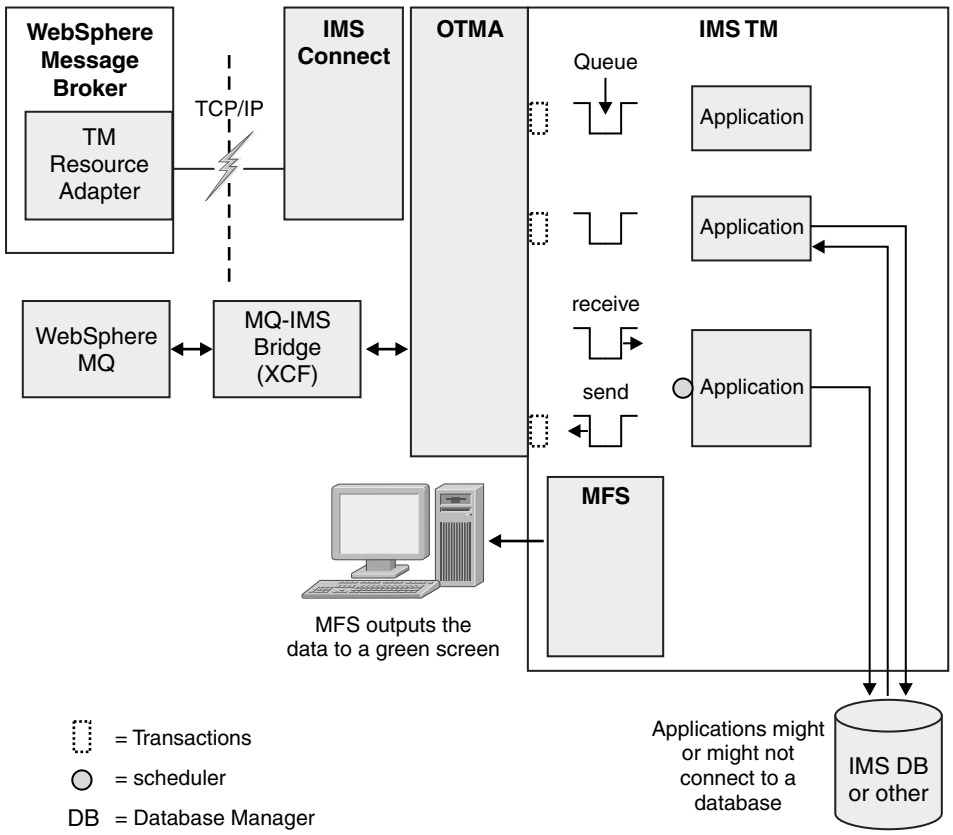
IMS includes two components:

IMS Database Manager (IMS DB)
A database management system for defining database structure, organizing business data, performing queries against the data, and performing database transactions.

IMS Transaction Manager (IMS TM)
A message-based transaction manager for processing input and output messages. IMS TM manages message queuing, security, scheduling, formatting, logging, and recovery.

In addition to these components, IMS Connect manages communications for IMS, connecting one or more clients with one or more IMS systems. IMS Connect also handles workload balancing, and supports the IBM supplied client, the IMS TM resource adapter.

The following diagram shows the layers of communication between WebSphere Message Broker and IMS.



You can also use the following methods to connect to an IMS system:

- **The WebSphere MQ-IMS bridge**

  The WebSphere MQ-IMS bridge is a component of WebSphere MQ for z/OS. You can use it to access applications on your IMS system from WebSphere MQ applications. For more information about the WebSphere MQ-IMS bridge, see WebSphere MQ Version 7 Information Center online.
- **The IMS SOAP Gateway**

  The IMS SOAP Gateway is a Web service solution that integrates IMS assets in a Service-Oriented Architecture (SOA) environment. For more information, see IMS SOAP Gateway Web page.

For more details about how IMS works with WebSphere Message Broker, see "IMS nodes."

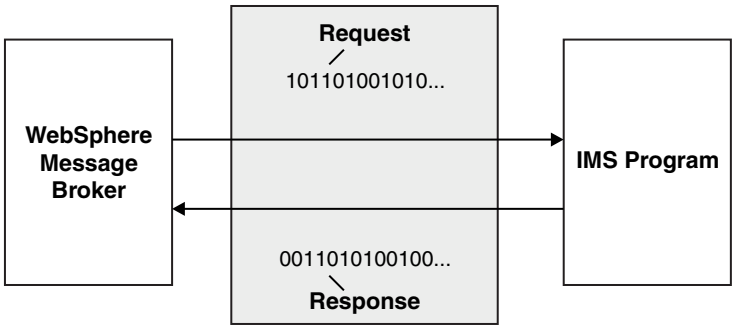
For further information about IMS and its components, see the IBM Information Management Software Library Web page, which contains links to the IMS information center and associated IBM Redbooks publications.

IMS nodes

WebSphere Message Broker message flows use IMS nodes to call programs that are running in IMS.

To enable function that becomes available in WebSphere Message Broker fix packs, use the **-f** parameter on the mqsichangebroker command. For more information, see mqsichangebroker command.

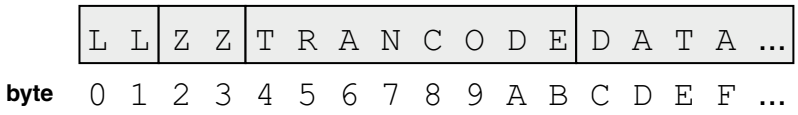
The IMS node sends a bit stream to IMS, which schedules one of its programs to process the message. The program generates a message, which IMS sends back to the IMS node, as illustrated in the following diagram.



The bit stream contains the routing information that IMS needs so that it can schedule a program to receive that bit stream. The structure of the bit stream varies depending on whether it is a request or response bit stream. The structure of the different bit streams are described in the following sections.

Request bit stream

The structure of the request bit stream is illustrated by the following diagram.

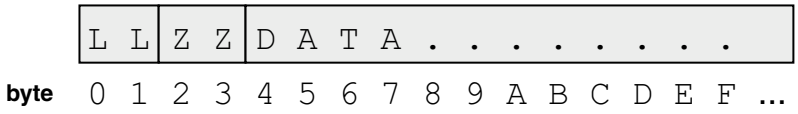


- *LLZZ* is a four-byte field. The first two bytes indicate the length of the bit stream, and the other two bytes are reserved for use by IMS.
- The transaction code can contain up to eight characters. If the code contains less than eight characters, the transaction code must be delimited by a space. When the transaction code is less than eight bytes, IMS reads only the transaction code and one space. The response segments do not need to have the transaction name, but an IMS program can add it.
- The rest of the bit stream comprises the data that the IMS program needs.

IMS reads the first twelve bytes of the bit stream, but it passes the entire bit stream to the IMS program.

Response bit stream

The structure of the response bit stream is illustrated by the following diagram.



Commands

You can also use bit streams to run commands. The structure of the response bit stream is illustrated by the following diagram.

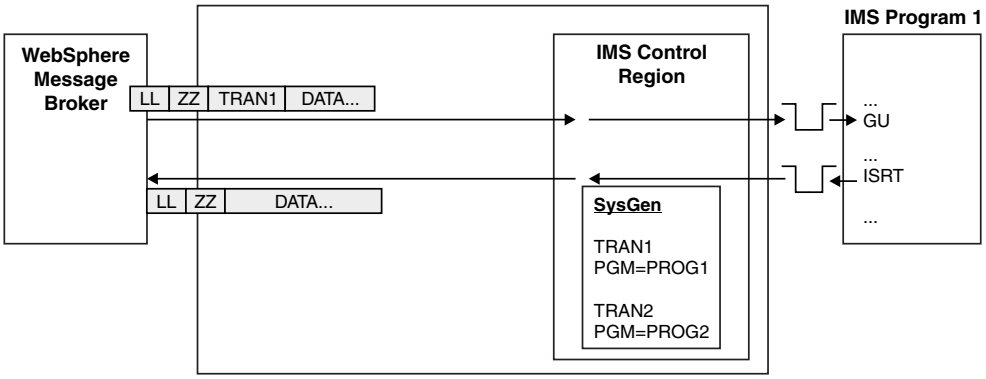
	L	L	Z	Z	/	D	I	S		T	R	A	N		A	L	L
byte	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	...

The first character after LLZZ is the slash (/) character, which is followed by the command verb and any arguments. For commands, the response bit stream has the same format as the response bit stream for transactions: LLZZ is followed by the response data.

For more information about IMS concepts, see the following topics:
- "IMS transactions and programs"
- "Response models"
- "IMS message structure" on page 63
- "IMS connections" on page 65

IMS transactions and programs:

The IMS system administrator defines the transactions. For each transaction that is defined, a program name is specified. When you invoke a transaction by using an IMS node, the IMS Control Region determines which program is configured for that transaction, and queues the data for retrieval by that program.



After the program has prepared the response data for the IMS node in the message flow, it inserts that data onto another queue. This output queue is tied to the socket to which WebSphere Message Broker is connected. Therefore, multiple concurrent message flows that are calling the same transaction each have a separate queue to receive the responses.

The IMS program gets messages by issuing a GU (GetUnique) call and it produces messages by issuing an ISRT (Insert) call. These calls are known as *DL/1 calls*. DL/1 is the programming interface to IMS. Other common DL/1 calls are PURG (purge) and GN (GetNext).

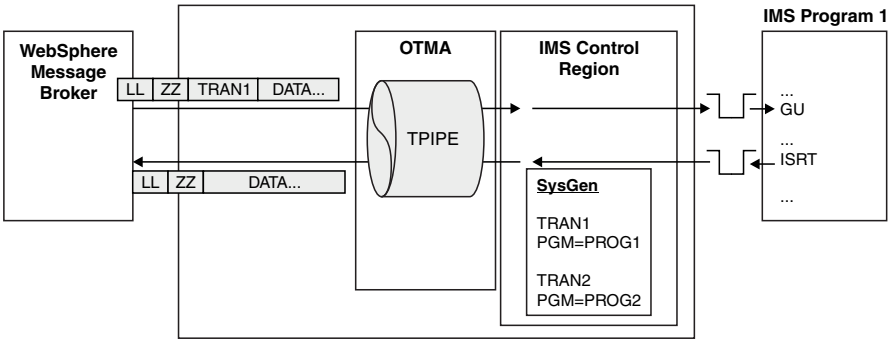
Response models:

The synchronous request and response model is associated with IMS.

Synchronous request and response

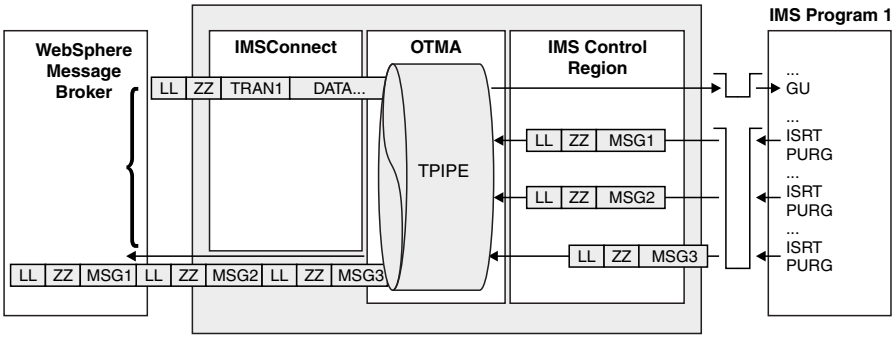
In the synchronous request and response model, the entire transmission is returned to the IMS node synchronously.

After the IMS program has prepared the response data for the WebSphere Message Broker message flow, it inserts that data onto a queue. WebSphere Message Broker uses Open Transaction Manager Access (OTMA) to communicate with the IMS program. OTMA helps to correlate the input to the IMS program with the output from the IMS program by using a transaction pipe (TPIPE). Therefore, multiple concurrent message flows that are calling the same transaction each receive the relevant response.



The TPIPE is created automatically. It is not necessary for the name of the TPIPE to be known to WebSphere Message Broker because it uses a mode of operation known as *sharable persistent sockets*, whereby a TPIPE is created automatically for every TCP/IP connection.

The output of a transaction can contain multiple messages. The IMS program inserts then purges each message, as shown in the following diagram. (For multi-segment output, the IMS program inserts multiple messages without purging them.) If the IMS program inserts multiple messages in a single sync point, OTMA correlates all of those messages to the input message and returns them all to the message flow as a single transmission.



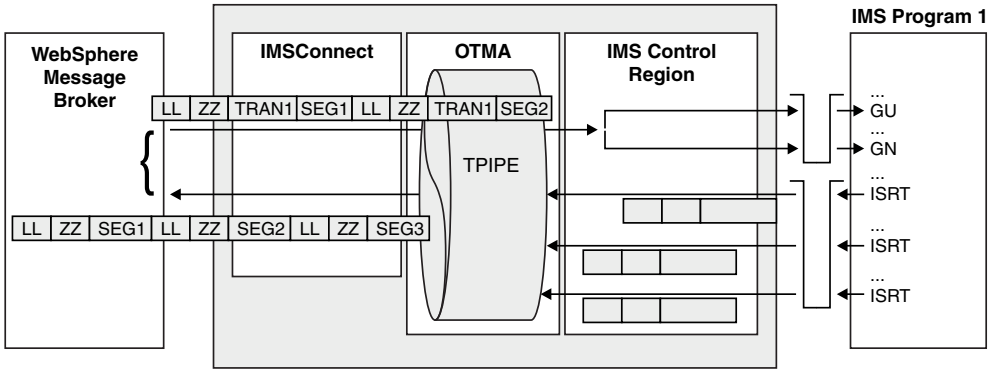
IMS message structure:

Each message that is sent to and from IMS can consist of one or more segments. IMS messages often contain multiple segments.

The bit stream that flows between WebSphere Message Broker and the IMS program (also known as the *transmission*) can contain multiple segments. Each segment begins with the LLZZ and Transaction code fields that are described in "IMS nodes" on page 60. The transmission can contain multiple messages, each one containing multiple segments. The IMS program gets the segments one at a

time and typically inserts the output data onto the queue one segment at a time. The IMS program purges the end of a message before it sends the first segment of the next message.

For input messages, each segment includes the LLZZ field. Only the first segment contains the transaction code (Trancode) field. For output messages, each segment includes the LLZZ field. The IMS program gets the segments one at a time. It makes a GetUnique (GU) call to read the first segment of the next message, and a GetNext (GN) call to read the next segment of the current message. The IMS program typically inserts the output data to the queue one segment at a time, and purges the end of a message before it sends the first segment of the next message, as shown in the following diagram.



A COBOL IMS program typically includes a copybook with the data structure definition of each segment. The program logic indicates the order in which the segments are retrieved and emitted by the program. The WebSphere Message Broker application has two ways to implement this information:

- Model the entire message in MRM.
- Model each segment in MRM but configure the message flow to reflect the application logic in determining the order of these segments.

An IMS transaction's response can have various structures:

- An MRM-CWF structure, such as a message definition that is derived from a COBOL copybook
- An MRM-TDS structure when the output is 3270–based, such as a list of name=value strings
- Another structure, such as XML output from a Java program that is running in an IMS Java Processing Region (JPR)

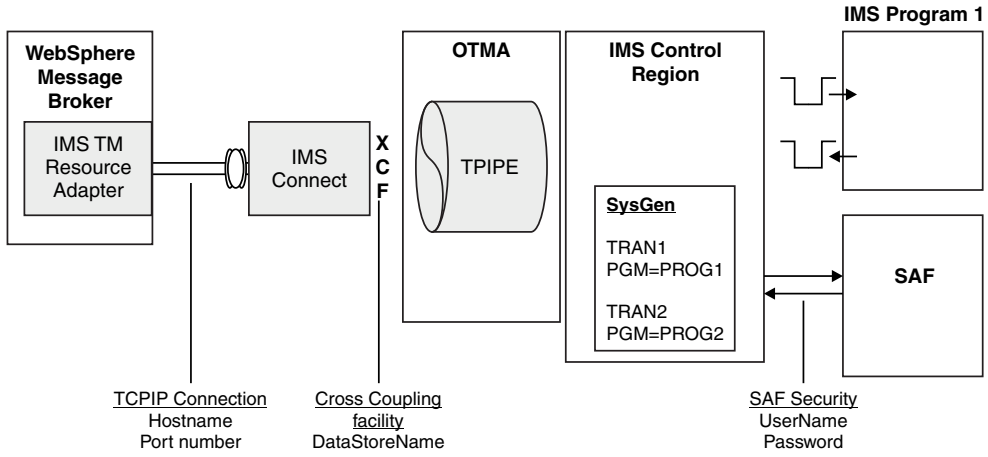
If the message definition is derived from a COBOL copybook, a message is a sequence of segments, each of which has a model built by importing its copybook. If the output is 3270-based, each segment is a line of output with an MRM-TDS model built by understanding the IMS transaction program's output.

IMS presents program output as one or more messages (typically, one output message per input message), each of which comprises one or more segments. The IMSRequest node presents the message as a single BLOB. You can parse the message into segments and use Filter or Compute nodes to test the shape of the response to determine how to re-parse the segments with ResetContentDescriptor nodes.

You must set the LL and ZZ values on output. The LL value is the entire length of the segment, including the four-byte LLZZ prefix. Therefore, the message flow typically requires an ESQL expression to calculate the LL value. The LLZZ field must use a big-endian encoding 785.

IMS connections:

Open Transaction Manager Access (OTMA) is used to provide access to IMS from WebSphere Message Broker.



OTMA uses the z/OS Cross Coupling Facility (XCF) to provide access to IMS from an OTMA client. To access a service through XCF, you must specify a data store name on the IMS node.

IMS Connect is an OTMA client that exposes a TCP/IP interface. WebSphere Message Broker uses the IMS TM Resource Adapter, which connects to IMS through IMS Connect. To connect to IMS Connect, you must specify a host name and port number. If the IMS system is configured to authenticate users by using a System Authorization Facility (SAF) product, such as RACF®, you must specify a user ID and password. You can use the mqsisetdbparms command to set a user ID and password for an IMSConnect configurable service. For detailed information about how to configure IMS Connect for security, see the *IMS Connect Security Support* topic in the IBM Information Management Software for z/OS Solutions Information Center.

Some restrictions exist for the OTMA environment that affect the range of IMS applications with which WebSphere Message Broker can interact. For example, OTMA cannot update the IMS main storage database (MSDB) because it has read only access to the database. For detailed information about the restrictions for the OTMA environment, see the *OTMA restrictions* topic in the IBM Information Management Software for z/OS Solutions Information Center.

Using configurable services for IMS nodes

You can configure IMS nodes to get connection details from a configurable service. For details about creating, changing, reporting, and deleting the configurable services, see “Changing connection information for the IMSRequest node” on page 223.

Configurable services

Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies; for example, an SMTP server or a JMS provider.

Instead of defining properties on the node or message flow, you can create configurable services so that nodes and message flows can refer to them to find properties at run time. If you use this method, you can change the values of attributes for a configurable service on the broker, which then affects the behavior of a node or message flow without the need for redeployment.

Unless it is explicitly stated by the function that is using the configurable service, you need to stop and start the execution group for the change of property value to take effect.

Use the following commands to work with configurable services:

- Use the mqsicreateconfigurableservice command to create configurable services.
- Use the mqsideleteconfigurableservice command to delete configurable services.
- Use the mqsichangeproperties command to set attributes after you have created the configurable services.
- Use the mqsireportproperties command to report attributes.

For a full list of configurable services and their properties, see Configurable services properties.

Message flow version and keywords

When you are developing a message flow, you can define the version of the message flow as well as other key information that you want to be associated with it.

After the message flow has been deployed, you can view the properties of the message flow in the workbench. These properties include the deployment and modification dates and times (the default information that is displayed) as well as any additional version or keyword information that you have set.

You can define information to give details of the message flow that has been deployed; therefore, you can check that it is the message flow that you expect.

Version

You can set the version of the message flow in the Version property.

You can also define a default message flow version in the Default version tag of the message flow preferences. All new message flows that are created after this value has been set have this default applied to the Version property at the message flow level.

Keywords

Keywords are extracted from the compiled message flow (the `.cmf` file) rather than the message flow source (the `.msgflow` file). Not all the source properties are added to the compiled file. Therefore, add message flow keywords in only these places:

- The label property of a Passthrough node

- ESQL comments or string literals
- The Long Description property of the message flow

Any keywords that you define must follow certain rules to ensure that the information can be parsed. The following example shows some values that you might want to define in the Long Description property:

```
$MQSI Author=John Smith MQSI$
$MQSI Subflow 1 Version=v1.3.2 MQSI$
```

The following table contains the information that the workbench shows.

Message flow name	
Deployment Time	28-Aug-2004 15:04
Modification Time	28-Aug-2004 14:27
Version	v1.0
Author	John Smith
Subflow 1 Version	v1.3.2

In this display, the version information has also been defined using the Version property of the object. If the version information has not been defined using the property, it is omitted from this display.

If message flows contain subflows, you can embed keywords in each subflow.

Restrictions within keywords

Do not use the following characters within keywords because they cause unpredictable behavior:

```
^ $ . | \ < > ? + * = & [ ] ( )
```

You can use these characters in the values that are associated with keywords; for example:

- `$MQSI RCSVER=$id$ MQSI$` is acceptable
- `$MQSI $name=Fred MQSI$` is not acceptable

Message flow connections

A connection is an entity that connects an output terminal of one message flow node to an input terminal of another. It represents the flow of control and data between two message flow nodes.

Most nodes have one input terminal, and many have more than one output terminal. You can connect an output terminal to more than one target node, so that the same message can be processed in a number of different ways. For example, you might want to send the same information to more than one remote application for further processing.

The connections of the message flow, represented by black lines in the Message Flow editor view, determine the path that a message takes through the message flow. You can add bend points to the connection to alter the way in which it is displayed.

See “Bend points” for a description of bend points. See “Message flow node terminals” for a description of terminals.

Bend points

A bend point is a point that is introduced in a connection between two message flow nodes at which the line that represents the connection changes direction.

Use bend points to change the visual path of a connection to display node alignment and processing logic more clearly and effectively. Bend points have no effect on the behavior of the message flow; they are visual modifications only.

A connection is initially made as a straight line between the two connected nodes or brokers. Use bend points to move the representation of the connection, without moving its start and end points.

Message flow node terminals

A terminal is the point at which one node in a message flow is connected to another node.

Use terminals to control the route that a message takes, depending on whether the operation that is performed by a node on that message is successful. Terminals are wired to other node terminals by using message flow node connections to indicate the flow of control.

Every built-in node has a number of terminals to which you can connect other nodes. Input nodes (for example, MQInput) do not have input terminals; all other nodes have at least one input terminal through which to receive messages to be processed. Most built-in nodes have failure terminals that you can use to manage the handling of errors in the message flow. Most nodes have output terminals through which the message can flow to a subsequent node.

If you have any user-defined nodes, these might also have terminals that you can connect to other built-in or user-defined node terminals.

Dynamic terminals are terminals that you can add to certain nodes after you have added them to a message flow in the Message Flow editor. For example, you can add dynamic output terminals to the PHPCompute, Route, and DatabaseRoute nodes, or you can add dynamic input terminals to the Collector node. You can also delete and rename dynamic terminals. If a node has five or more terminals, they are displayed in a group. For example, the following example shows a node with more than four output nodes. 

Threading

A message flow is inherently thread-safe, and message flows can be run concurrently on more than one thread.

Each message passing through a message flow for processing by a series of nodes executes on a single thread. If you want to increase the throughput of a message flow, you can increase the number of threads assigned to that message flow. With a larger number of threads, the message flow can handle peak message loads. At other times, the additional threads are idle.

You can increase or decrease the number of threads servicing a flow by using the Additional instances property on the input node of the message flow.

Each instance of a message flow processing node is shared, and used by all the threads that service the message flow in which the node is defined.

Execution and threading models in a message flow

The execution model is the system used to start message flows which process messages through a series of nodes.

When an execution group is initialized, the appropriate loadable implementation library (LIL) files and Plug-in Archive (PAR) files are made available to the runtime environment. The execution group runtime process starts, and creates a dedicated configuration thread.

The message flow execution environment is conceptually like procedural programming. Nodes that you insert into a message flow are like subroutines that are called by using a function call interface. However, rather than a call-return interface, in which parameters are passed in the form of input message data, the execution model is referred to as a propagation-and-return model.

In the message flow execution environment, the message flow is thread-safe. You can run message flows concurrently on many operating system threads, without having to consider serialization issues.

Each input message that passes through a message flow for processing by a series of nodes executes on a single thread; it is processed only by the thread that received it. If you want to increase the throughput of a message flow, you can increase the number of threads that are assigned to that message flow. The memory requirements of an execution group are not unduly affected by running message flows on more operating system threads.

With a larger number of threads, the message flow can handle peak message loads. At other times, the additional threads are idle.

You can increase or decrease the number of threads servicing a flow by using the Additional instances property on the input node of the message flow.

Each instance of a message flow processing node is shared, and used by all the threads that service the message flow in which the node is defined.

The message tree

A message tree is a structure that is created, either by one or more parsers when an input message bit stream is received by a message flow, or by the action of a message flow node.

A message is used to describe:

- A set of business data that is exchanged by applications
- A set of elements that are arranged in a predefined structure
- A structured sequence of bytes

WebSphere Message Broker routes and manipulates messages after converting them into a logical tree. The process of conversion, called parsing, makes obvious

the content and structure of a message, and simplifies later operations. After the message has been processed, the parser converts it back into a bit stream.

WebSphere Message Broker supplies a range of parsers to handle the many different messaging standards in use. See Parsers.

After a message has been parsed, it can be processed in a message flow.

The logical tree has contents that are identical to the message, but the logical tree is easier to manipulate in the message flow. The message flow nodes provide an interface to query, update, or create the content of the tree.

How the message tree is populated

The message tree is initially populated by the input node of the message flow.

When the input node receives the input message, it creates and completes the Properties tree (the first subtree of the message tree). See "Message tree structure" on page 77.

The node then examines the contents of the input message bit stream and creates the remainder of the message tree to reflect those contents. This process depends to some extent on the input node itself, which is governed by the transport across which the message is received:

WebSphere MQ Enterprise Transport and WebSphere MQ Telemetry Transport protocols

If your application communicates with the broker across these protocols, and your message flow includes the corresponding MQInput or SCADAInput node, all messages that are received must start with a Message Queue Message Descriptor (MQMD) header. If a valid MQMD is not present at the start of the message, the message is rejected, and no further processing takes place.

The input node first invokes the MQMD parser and creates the subtree for that header.

A message can have zero or more additional headers following the MQMD. These headers are chained together, with the Format field of one header defining the format of the following header, up to and including the last header, which defines the format of the message body. If an MQRFH and an MQRFH2 header exist in the chain, the name and value data in either of these two headers can also contain information about the format of the following data. If the value that is specified in Format is a recognized parser, this value always takes precedence over the name and value data.

The broker invokes the appropriate parser to interpret each header, following the chain in the message. Each header is parsed independently. The fields in a single header are parsed in an order that is governed by the parser. You cannot predict the order that is chosen, but the order in which fields are parsed does not affect the order in which the fields are displayed in the header.

The broker ensures that the integrity of the headers that precede a message body is maintained. The format of each part of the message is defined, either by the Format field in the immediately preceding header (if the following part is a recognized WebSphere MQ format), or by the values that are set in the MQRFH or MQRFH2 header:

- The format of the first header is known because it must be MQMD.
- The format of any subsequent header in the message is set in the Format field in the preceding header.
- The format of the body corresponds to the message domain and the parser that must be invoked for the message body (for example, XMLNSC). This information is set either in the MQRFH or MQRFH2 header, or in the *Message Domain* property of the input node that receives the message.

This process is repeated as many times as required by the number of headers that precede the message body. You do not need to populate these fields yourself; the broker handles this sequence for you.

The broker completes this process to ensure that Format fields in headers correctly identify each part of the message. If the broker does not complete this process, WebSphere MQ might be unable to deliver the message. The message body parser is not a recognized WebSphere MQ header format, therefore the broker replaces this value in the last header's Format field with the value MQFMT_NONE. The original value in that field is stored in the Domain field in the MQRFH or MQRFH2 header to retain the information about the contents of the message body.

For example, if the MQRFH2 header immediately precedes the message body, and its Format field is set to XMLNSC, which indicates that the message body must be parsed by the XMLNSC parser, the MQRFH2 Domain field is set to XMLNSC, and its Format field is reset to MQFMT_NONE.

These actions might result in information that is stored explicitly by an ESQL or Java expression being replaced by the broker.

When all the headers have been parsed, and the corresponding sub-trees have been created in the message tree, the input node associates the specified parser with the message body. Specify the parser that is to be associated with the message body content, either in a header in the message (for example, the <mcd> folder in the MQRFH2 header), or in the input node properties (if the message does not include headers). The input node makes the association as described in the following list:

- If the message has an MQRFH or MQRFH2 header, the domain that is identified in the header (either in Format or the name and value data) determines the parser that is associated with this message.

  The SCADAInput node creates WebSphere MQ format messages with MQRFH2 headers from the input messages that the listener receives on the TCP/IP port.
- If the message does not have an MQRFH or MQRFH2 header, or if the header does not identify the domain, the *Message Domain* property of the input node indicates the domain of the message, and the parser that is to be used. You can specify a user-defined domain and parser.
- If the message domain cannot be identified by header values or by the *Message Domain* property of the input node, the message is handled as a binary object (BLOB). The BLOB parser is associated with the message. A BLOB can be interpreted as a string of hexadecimal characters, and can be modified or examined in the message flow by specifying the location of the subset of the string.

By default, the message body is not parsed straight away, for performance reasons. The message body might not need to be parsed during the message flow. It is parsed only when a reference is made to its contents.

For example, the message body is parsed when you refer to a field in the message body, for example: `Root.XMLNSC.MyDoc.MyField`. Depending on the paths that are taken in the message flow, this parse can take place at different points. This "parse when first needed" approach is also referred to as 'partial parsing' or 'on-demand parsing', and in typical processing does not affect the logic of a message flow. However, there are some implications for error handling scenarios; see “Handling errors in message flows” on page 244.

If you want a message flow to accept messages from more than one message domain, include an MQRFH2 header in your message from which the input nodes extract the message domain and related message definition information (message set, message type, and message format).

If you set up the message headers or the input node properties to identify a user-defined domain and parser, the way in which it interprets the message and constructs the logical tree might differ from that described here.

WebSphere MQ Multicast Transport, WebSphere MQ Real-time Transport, WebSphere Broker File Transport, WebSphere Broker Adapters Transport, WebSphere MQ Web Services Transport, and WebSphere Broker JMS Transport protocols

If your application communicates with the broker across these supported protocols, and your message flow includes the corresponding input nodes, messages that are received do not have to include a particular header. If recognized headers are included, the input node invokes the appropriate parsers to interpret the headers and to build the relevant parts of the message tree, as described for the other supported protocols.

If there are no headers, or these headers do not specify the parser for the message body, set the input node properties to define the message body parser. If you do not set the node properties in this way, the message is treated as a BLOB. You can specify a user-defined parser.

The specified parser is associated with the message body by the input node (in the same way as it is for the WebSphere MQ Enterprise Transport and WebSphere MQ Telemetry Transport protocols), and by default the message body is not parsed immediately.

If you set up the message headers or the input node properties to identify a user-defined domain and parser, the way in which it interprets the message and constructs the logical tree might differ from that described here.

All other protocols

If you want your message flow to accept messages from a transport protocol for which WebSphere Message Broker does not provide built-in support, or you want it to provide some specific processing on receipt of a message, use either the Java or the C language programming interface to create a new user-defined input node.

This interface does not automatically generate a Properties subtree for a message (this subtree is discussed in “Message tree structure” on page 77). A message does not need to have a Properties subtree, but you might find it useful to create one to provide a consistent message tree structure,

regardless of input node. If you are using a user-defined input node, you must create a Properties subtree in the message tree yourself.

To process messages that do not conform to any of the defined message domains, use the C language programming interface to create a new user-defined parser.

Refer to the node interface to understand how it uses parsers, and whether you can configure it to modify its behavior. If the node uses a user-defined parser, the tree structure that is created for the message might differ slightly from that created for built-in parsers. A user-defined input node can parse an input message completely, or it can participate in partial parsing in which the message body is parsed only when it is required.

You can also create your own output and message processing nodes in C or Java.

Properties versus MQMD folder behavior for various transports

Differences exist in the way the Properties folder and the MQMD folder are treated with respect to which folder takes precedence for the same fields. This treatment is characterized by the transport type (for example, HTTP or WebSphere MQ) that you use.

When the message flow is sourced by an MQInput node, you have an MQMD to parse. In this case, the Properties folder is sourced by the MQMD values and so the MQMD folder takes precedence over the Properties folder in terms of value propagation between the folders. This scenario means that you can perform ESQL, for example, `SET OutputRoot.MQMD.CorrelId` and this command updates the Properties folder value.

When the message flow is sourced from a input node that is not the MQInput node (such as the HTTPInput node or a user-defined input node), no MQMD is parsed. In this scenario, the Properties folder is not sourced from an input MQMD; it is created and populated with transport values that come from other transport specific headers. When you create an MQMD folder in a message flow that was not sourced from the WebSphere MQ transport, the MQMD header does not take precedence over the Properties folder because the WebSphere MQ transport did not populate the Properties folder. Therefore, in this case, the Properties folder overwrites any values in MQMD.

The Properties folder is constructed and represents a message received on the transport. In this scenario two entirely different transports are being used which have different meanings and, therefore, different requirements of the Properties folder. When sourced from an HTTPInput node, the HTTP headers have control over the Properties folder for like fields. When sourced from an MQInput node the MQMD has control over the Properties folder for like fields.

Therefore, when you add an MQMD folder to a tree that was created by the HTTP Transport, this MQMD folder does not have control over the Properties folder, and the value propagation direction is not MQMD to Properties, it is Properties to MQMD. The correct approach is to set the replyIdentifier field of the Properties folder and to use it to populate the MQMD:

```
SET OutputRoot.Properties.ReplyIdentifier = X' ..... ';
```

The behavior is not unique to just the CorrelId to ReplyIdentifier fields. It applies for all like fields between the MQMD and Properties folder:

- CorrelId
- Encoding
- CodedCharSetId
- Persistence
- Expiry
- Priority

In summary:

1. When your message flow is sourced by an MQInput node, the MQMD takes precedence over the Properties folder in terms of value propagation between the folders.
2. When your message flow is sourced from an input node that is not the MQInput node (such as the HTTPInput node or a user-defined input node), the MQMD header does not take precedence over Properties folder .
3. When a MQMD folder is added in a tree that was created by the HTTP Transport, this MQMD does not have control over the Properties folder and the value propagation direction is not MQMD to Properties; it is Properties to MQMD.

Example

```
SET OutputRoot.Properties = InputRoot.Properties;
SET OutputRoot.MQMD.Version = 2;
SET OutputRoot.MQMD.CorrelId = X'4d454e53414a4533202020202020202020202020202020';
SET OutputRoot.MQMD.Encoding = 785;
SET OutputRoot.MQMD.CodedCharSetId = 500;
SET OutputRoot.MQMD.Persistence = 1;
SET OutputRoot.MQMD.Expiry = 10000;
SET OutputRoot.MQMD.Priority = 9;
SET OutputRoot.BLOB.BLOB = X'01';
```

When sourced from an HTTPInput node none of these changes will take effect and the MQMD output from the Compute node is:

```
(0x01000000):MQMD       = (
    (0x03000000):Version        = 2
    (0x03000000):CorrelId       = X'000000000000000000000000000000000000000000000000'
    (0x03000000):Encoding       = 546
    (0x03000000):CodedCharSetId = 1208
    (0x03000000):Persistence    = FALSE
    (0x03000000):Expiry         = -1
    (0x03000000):Priority       = 0
```

Message tree contents after an exception:

The contents of the message tree are updated if an exception is raised.

If no exception occurs while processing the message, the tree structure and content received by an individual node is determined by the action of previous nodes in the flow. If an exception occurs in the message flow, the content of the four trees depends on the following factors:

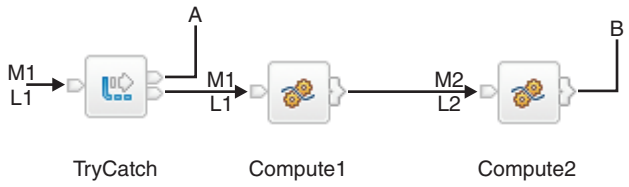
- If the exception is returned to the input node, and the input node Catch terminal is not connected, the trees are discarded. If the message is in a transaction, it is returned to the input queue for further processing. When the message is processed again, a new tree structure is created. If the message is not in a transaction, it is discarded.
- If the exception is returned to the input node and the Catch terminal is connected, the message and local environment trees that were created originally

by the input node, and propagated through the Out terminal, are restored, and any updates that you made to their content in the nodes that followed the input node are lost. The environment tree is not restored, and its contents are preserved. If the nodes following the input node include a Compute node that creates a new local environment or message tree, those trees are lost. The exception list tree reflects the one or more exceptions that have been recorded.

- If the exception is caught in the message flow by a TryCatch node, the message and local environment trees that were previously propagated through the Try terminal of the TryCatch node are restored and propagated through the Catch terminal. Any updates that you made to their content in the nodes that followed the TryCatch node are lost. The environment tree is not restored, and its contents are preserved. If the nodes following the TryCatch node include a Compute node that creates a new local environment or message tree, those trees are lost. The exception list tree reflects the one or more exceptions that have been recorded.

Exception handling paths in a message flow: Exception handling paths start at a failure terminal (most message processing nodes have these), the Catch terminal of an input node, a TryCatch node, or an AggregateReply node, but are no different in principle from a normal message flow path. Such a flow consists of a sequence of nodes connected together by the designer of the message flow. The exception handling paths differ in the kind of processing that they do to record or react to the exception. For example, they might examine the exception list to determine the nature of the error, and take appropriate action or log data from the message or exception.

The local environment and message tree that are propagated to the exception handling message flow path are those at the start of the exception path, not those at the point when the exception is thrown. The following figure illustrates this point:



- A message (M1) and local environment (L1) are being processed by a message flow. They are passed through the TryCatch node to Compute1.
- Compute1 updates the message and local environment and propagates a new message (M2) and local environment (L2) to the next node, Compute2.
- An exception is thrown in Compute2. If the failure terminal of Compute2 is not connected (point **B**), the exception is propagated back to the TryCatch node, but the message and local environment are not. The exception handling path starting at point **A** has access to the first message and local environment, M1 and L1. The environment tree is also available and retains the content it had when the exception occurred.
- If the failure terminal of Compute2 is connected (point **B**), the message and local environment M2 and L2 are propagated to the node connected to that failure terminal. The environment tree is also available and retains the content it had when the exception occurred.

Logical tree structure

The logical tree structure is the internal (broker) representation of a message. It is also known as the message assembly.

When a message arrives at a broker, it is received by an input node that you have configured in a message flow. Before the message can be processed by the message flow, the message must be interpreted by one or more parsers that create a logical tree representation from the bit stream of the message data.

The tree format contains identical content to the bit stream from which it is created, but it is easier to manipulate in the message flow. Many of the built-in message flow nodes provide an interface for you to query and update message content in the tree, and perform other actions against messages and databases to help you to provide the required function in each node.

Several interfaces are provided:

- ESQL, a programming language that you can code in the Compute, Database, and Filter nodes.
- Java, a programming language that you can code in the JavaCompute node.
- Mappings, a graphical method of achieving transformation from input to output structures, available in the DataDelete, DataInsert, DataUpdate, Mapping, and Warehouse nodes.
- XSL, a language for transforming XML that you can use in the XSLTransform node.
- PHP, a scripting language that you can code in the PHPCompute node.

The tree structure that is created by the parsers is largely independent of any message format (for example, XML). The exception to this is the subtree that is created as part of the message tree to represent the message body. This subtree is message dependent, and its content is not further described here.

The input node creates this message assembly, which consists of four trees:

- "Message tree structure" on page 77
- "Environment tree structure" on page 79
- "Local environment tree structure" on page 80
- "Exception list tree structure" on page 85

The first of these trees is populated with the contents of the input message bit stream, as described in "How the message tree is populated" on page 70; the remaining three trees are initially empty.

Each of the four trees created has a root element (with a name that is specific to each tree). Each tree is made up of a number of discrete pieces of information called *elements*. The root element has no *parent* and no *siblings* (siblings are elements that share a single parent). The root is parent to a number of *child* elements. Each child must have a parent, can have zero or more siblings, and can have zero or more children.

The four trees are created for both built-in and user-defined input nodes and parsers.

The input node passes the message assembly that it has created to subsequent message processing nodes in the message flow:

- All message processing nodes can read the four trees.

- You can code ESQL in the Database and Filter nodes, or use mappings in the nodes that support that interface to modify the Environment and LocalEnvironment trees only.
- The Compute node differs from other nodes in that it has both an input message assembly and at least one output message assembly. Configure the Compute node to determine which trees are included in the output message assembly; the Environment tree is an exception in that it is always retained from input message assembly to output message assembly.

  To determine which of the other trees are included, you must specify a value for the Compute mode property of the node (displayed on the Advanced tab). The default action is for only the message to be created. You can specify any combination of message, LocalEnvironment, and ExceptionList trees to be created in the output message assembly.

  If you want the output message assembly to contain a complete copy of the input message tree, you can code a single ESQL SET statement to make the copy. If you want the output message to contain a subset of the input message tree, code ESQL to copy those parts that you want. In both cases, your choice of Compute mode must include `Message`.

  If you want the output message assembly to contain all or part of the input LocalEnvironment or ExceptionList tree contents, code the appropriate ESQL to copy information you want to retain in that tree. Your choice of Compute mode must include LocalEnvironment, or Exception, or both.

  You can also code ESQL to populate the output message, Environment, LocalEnvironment, or ExceptionList tree with information that is not copied from the input tree. For example, you can retrieve data from a database, or calculate content from the input message data.
- You can produce similar results in the JavaCompute node. See “Writing Java” on page 526 for more information.
- You can also achieve results like these in the PHPCompute node. See “Using PHP” on page 455 for more information.

Message tree structure:

The message tree is a part of the logical message tree in which the broker stores its internal representation of the message body.

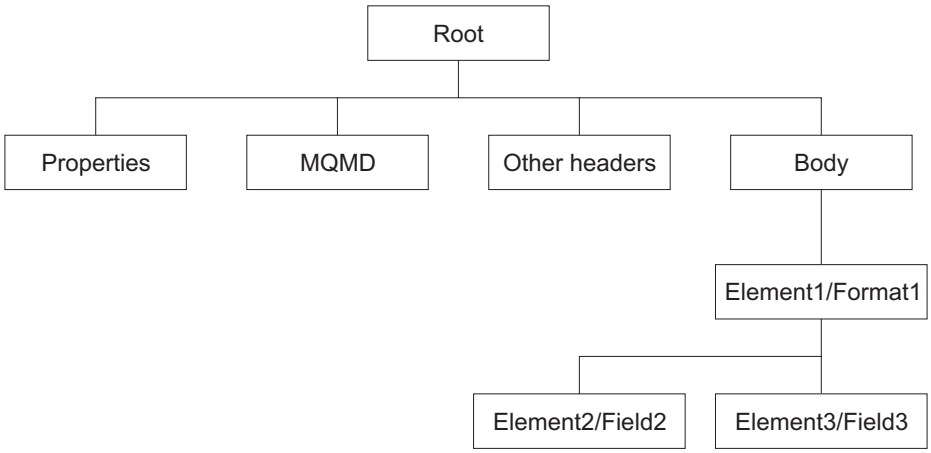
The root of a message tree is called Root. The message tree is always present, and is passed from node to node in a single instance of a message flow.

The message tree includes all the headers that are present in the message, in addition to the message body. The tree also includes the Properties subtree (described in “Parsers” on page 91), if that is created by the parser. If a supplied parser has created the message tree, the element that represents the Properties subtree is followed by zero or more headers.

If the message has been received across the WebSphere MQ Enterprise Transport, WebSphere MQ Mobile Transport, or WebSphere MQ Telemetry Transport, the first header (the second element) must be the MQMD. Any additional headers that are included in the message appear in the tree in the same order as in the message. The last element beneath the root of the message tree is always the message body.

If a user-defined parser has created the message tree, the Properties tree, if present, is followed by the message body.

The message tree structure is shown in the following section. If the input message is not a WebSphere MQ message, the headers that are shown might not be present. If the parser that created this tree is a user-defined parser, the Properties tree might not be present.



The Body tree is a structure of child elements that represents the message content (data), and reflects the logical structure of that content. The Body tree is created by a body parser (either a supplied parser or a user-defined parser), as described in "Parsers" on page 91.

Each element in the parsed tree is one of three types:

Name element
A name element has a string associated with it, which is the name of the element. An example of a name element is `XMLElement`, as described in "XML element" on page 1572. A name element also has a second string associated with it, which is the namespace of the element; this string might be empty.

Value element
A value element has a value associated with it. An example of a value element is `XMLContent`, as described in "XML content" on page 1572.

Name-value element
A name-value element is an optimization of the case where a name element contains only a value element and nothing else. The element contains both a name and a value. An example of a name-value element is `XMLAttribute`, as described in "XML attribute" on page 1570.

For information about how the message tree is populated, see "How the message tree is populated" on page 70.

Properties folder: The Properties folder is the first element of the message tree and holds information about the characteristics of the message.

The root of the Properties folder is called Properties. It is the first element under Root. All message trees that are generated by the built-in parsers include a Properties folder for the message. If you create your own user-defined parser, you can choose whether the parser creates a Properties folder. However, for consistency, you should include this action in the user-defined parser.

The Properties folder contains a set of standard properties that you can manipulate in the message flow nodes in the same way as any other property. Some of these

fields map to fields in the supported WebSphere MQ headers, if present, and are passed to the appropriate parser when a message is delivered from one node to another.

For example, the MQRFH2 header contains information about the message set, message type, and message format. These values are stored in the Properties folder as MessageSet, MessageType, and MessageFormat. To access these values using ESQL or Java in the message processing nodes, refer to these values in the Properties folder; do not refer directly to the fields in the headers from which they are derived.

The Properties parser ensures that the values in the header fields match the values in the Properties folder on input to, and output from, every node. For any field, if only one header is changed (the Properties header or a specific message header), that value is used. If both the Properties header and the specific message header are changed, the value from the Properties folder is used.

When the message flow processing is complete, the Properties folder is discarded.

Environment tree structure:

The environment tree is a part of the logical message tree in which you can store information while the message passes through the message flow.

The root of the environment tree is called Environment. This tree is always present in the input message; an empty environment tree is created when a message is received and parsed by the input node. You can use this tree as you choose, and create both its content and structure.

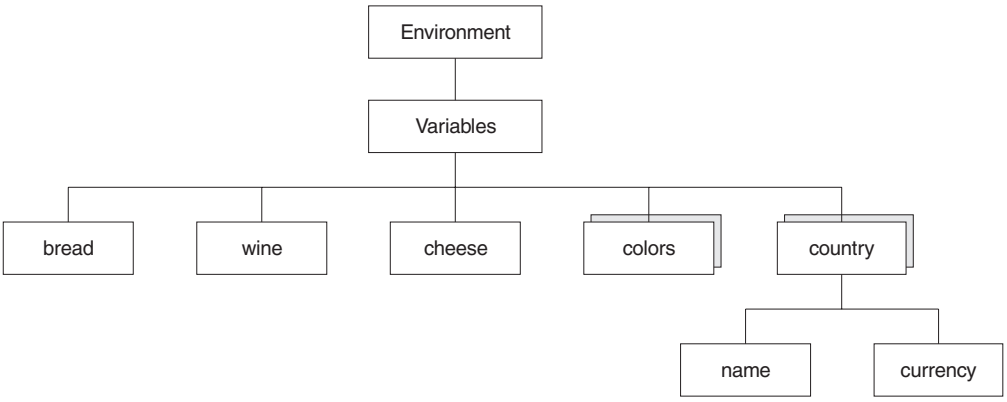
WebSphere Message Broker uses fields in the Environment tree in only two situations. (Contrast this with the "Local environment tree structure" on page 80, which the broker uses in many situations):

- If you have requested data collection for message flow accounting and statistics, and have indicated that accounting origin basic support is required, the broker checks for the existence of the field Environment.Broker.AccountingOrigin. If the field exists, the broker uses its value to set the accounting origin for the current data record. For further information about the use of this field, see "Setting message flow accounting and statistics accounting origin" on page 659.
- If you have activated a message flow to emit monitoring events the broker stores correlation attributes in the Environment tree. For further information, see "Correlation and monitoring events" on page 1497

The environment tree differs from the local environment tree in that a single instance of it is maintained throughout the message flow. If you include a Compute node, a Mapping node, or a JavaCompute node in your message flow, you do not have to specify whether you want the environment tree to be included in the output message. The environment tree is included automatically, and the entire contents of the input environment tree are retained in the output environment tree, subject to any modifications that you make in the node. Any changes that you make are available to subsequent nodes in the message flow, and to previous nodes if the message flows back (for example, to a FlowOrder or TryCatch node).

If you want to create your own information, create it in the environment tree in a subtree called Variables.

The following figure shown an example of an environment tree:



You could use the following ESQL statements to create the content shown above.

```
SET Environment.Variables =
      ROW('granary' AS bread, 'riesling' AS wine, 'stilton' AS cheese);
SET Environment.Variables.Colors[] =
      LIST{'yellow', 'green', 'blue', 'red', 'black'};
SET Environment.Variables.Country[] = LIST{ROW('UK' AS name, 'pound' AS currency),
     ROW('USA' AS name, 'dollar' AS currency)};
```

When the message flow processing is complete, the Environment tree is discarded.

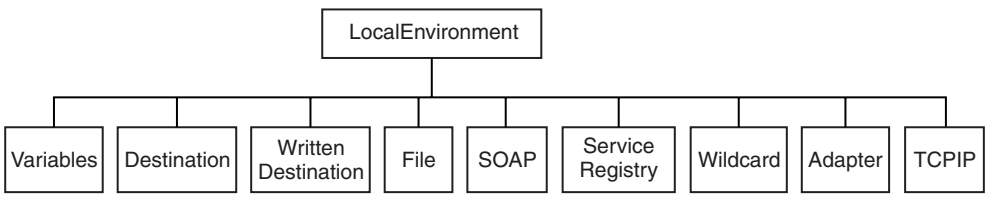
Local environment tree structure:

The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message.

The root of the local environment tree is called LocalEnvironment. This tree is always present in the input message, it is created when a message is received by the input node. Some input nodes create local environment fields, others leave it empty.

Use the local environment tree to store variables that can be referred to and updated by message processing nodes that occur later in the message flow. You can also use the local environment tree to define destinations (that are internal and external to the message flow) to which a message is sent. WebSphere Message Broker also stores information in LocalEnvironment in some circumstances, and references it to access values that you might have set for destinations. (Contrast this to the Environment tree structure, which the broker only uses in specific situations, see “Environment tree structure” on page 79.)

The following figure shows an example of the local environment tree structure. The children of Destination are protocol-dependent.



In the tree structure shown, LocalEnvironment has several children:

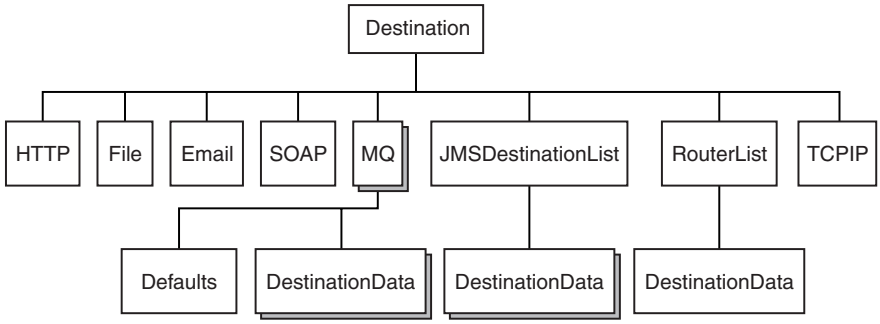
LocalEnvironment.Variables

This subtree is optional. If you create local environment variables, store them in a subtree called Variables. It provides a work area that you can use to pass information between nodes. This subtree is never inspected or modified by any supplied node.

Variables in the local environment can be changed by any subsequent message processing node, and the variables persist until the node that created them goes out of scope.

The variables in this subtree are persistent only within a single instance of a message flow. If you have multiple instances of a message passing through the message flow, and need to pass information between them, you must use an external database.

LocalEnvironment.Destination



This subtree consists of a number of children that indicate the transport types to which the message is directed (the Transport identifiers), or the target Label nodes that are used by a RouteToLabel node.

- Transport information

  Transport information is used by some input and output nodes, including HTTP, MQ, JMS, SOAP, File, E-mail, and TCPIP.

  LocalEnvironment.Destination.HTTP

  If the message flow starts with an HTTPInput node, a single name element HTTP is added to Destination. The element HTTP.RequestIdentifier is created and initialized so that it can be used by an HTTPReply node. You can also create other fields in the HTTP structure for use by the HTTPRequest node; for example, the URL of the service to which the request is sent. The topic for each node contains more information about the contents of Destination for the WebSphere MQ Web Services Transport protocol.

  LocalEnvironment.Destination.File

  If the message flow includes a FileOutput node, you can override its directory and name properties with elements in this subtree. See “Using local environment variables with file nodes” on page 849.

  LocalEnvironment.Destination.Email

  If the message flow includes an EmailOutput node, then information defined in this subtree will specify or override the SMTP server connection information and attachments associated with each e-mail sent by the node. Multiple attachments can be specified for inclusion in the e-mail sent, including the specification of the attachment name, content, and type. See “EmailOutput node” on page 933.

  LocalEnvironment.Destination.SOAP

You can place outbound WS-Addressing header information in the local environment to override the defaults that are generated by the SOAPReply, SOAPRequest, or SOAPAsyncRequest nodes. See "WS-Addressing information in the local environment" on page 756.

If the message flow includes SOAPRequest or SOAPAsyncRequest nodes, you can override their HTTP Transport properties in this subtree. See "SOAPRequest node" on page 1204, "SOAPAsyncRequest node" on page 1172, or "Local environment overrides for the SOAPRequest node" on page 1212.

If the message flow includes a SOAPAsyncRequest node, you can use this subtree to pass state and correlation information to a SOAPAsyncResponse node in another message flow. See "WS-Addressing with the SOAPAsyncRequest and SOAPAsyncResponse nodes" on page 754.

If the message flow includes SOAPReply, SOAPRequest, or SOAPAsyncRequest nodes, you can override their use of outbound MTOM messages in this subtree. See "Using SOAP MTOM with the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes" on page 748.

LocalEnvironment.Destination.MQ

If the message flow includes an MQOutput node, each element is a name element, *MQ* (A deprecated alternative exists, called *MQDestinationList*. Use *MQ* for all new message flows). If more than one element exists, each is processed sequentially by the node. See the example in "Populating Destination in the local environment tree" on page 355.

You can configure MQOutput nodes to examine the list of destinations and send the message to those destinations, by setting the property Destination Mode to Destination List. If you do so, you must create this subtree and its contents to define those destinations, giving it the name Destination. If you do not do so, the MQOutput node cannot deliver the messages.

If you prefer, you can configure the MQOutput node to send messages to a single fixed destination, by setting the property Destination Mode to Queue Name or Reply To Queue. If you select either of these fixed options, the destination list has no effect on broker operations and you do not have to create this subtree.

You can construct the MQ element to contain a single optional Defaults element. The Defaults element, if created, must be the first child and must contain a set of name-value elements that give default values for the message destination and its PUT options for that parent.

You can also create a number of elements called DestinationData within MQ. Each of these can be set up with a set of name-value elements that defines a message destination and its PUT options.

The set of elements that define a destination is described in "Data types for elements in the MQ DestinationData subtree" on page 1519.

The content of each instance of DestinationData is the same as the content of Defaults for each protocol, and can be used to override the default values in Defaults. You can set up Defaults to contain values that are common to all destinations, and set only the unique values in each DestinationData subtree. If you do not set a value either in DestinationData or Defaults, the value that you have set for the corresponding node property is used. Similarly, if you specify a field

name or value with the wrong spelling or case, it is ignored, and the value that you have set for the corresponding node property is used.

The information that you insert into DestinationData depends on the characteristic of the corresponding node property: this information is described in "Accessing the local environment tree" on page 353.

LocalEnvironment.Destination.JMSDestinationList

A JMSOutput node can be configured to send to multiple JMS Queues or to publish to multiple JMS Topics using a destination list created in the local environment by a transformation node.

The JMSOutput node searches the local environment for data elements called DestinationData under the folder Destination.JMSDestinationList. The node sends the JMS message to each DestinationData entry found in that folder. See the example in "Populating Destination in the local environment tree" on page 355.

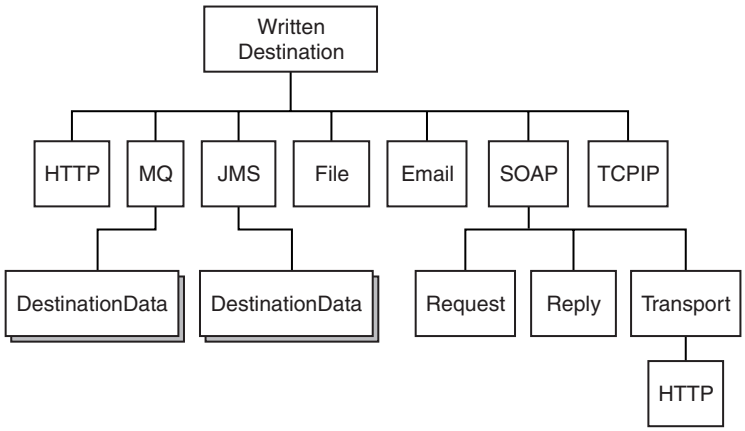
LocalEnvironment.Destination.TCPIP

If the message flow includes a TCPIPClientOutput node or a TCPIPServerOutput node, you can override its TCPIP connection with elements in this subtree. See "TCPIPClientOutput node" on page 1225 and "TCPIPServerOutput node" on page 1256.

- Routing information

  The child of Destination is *RouterList*. It has a single child element called *DestinationData*, which has a single entry called labelName. If you are using a dynamic routing scenario involving the RouteToLabel and Label nodes, you must set up the Destination subtree with a RouterList that contains the reference labels.

LocalEnvironment.WrittenDestination



This subtree contains the addresses to which the message has been written. Its name is fixed and it is created by the message flow when a message is propagated through the Out terminal of a request, output, or reply node. The subtree includes transport-specific information (for example, if the output message has been put to a WebSphere MQ queue, it includes the queue manager and queue names).

You can use one of the following methods to obtain information about the details of a message after it has been sent by the nodes:

- Connect a transformation node to the Out terminal.
- Configure a user exit to process an output message callback event, as described in "Exploiting user exits" on page 239.

The topic for each node that supports WrittenDestination information contains details about the data that it contains.

LocalEnvironment.File
This subtree contains information that is stored by the FileInput node.

This information describes the file, and also contains data about the current record.

More details about the information that is stored in this subtree are in "Using local environment variables with file nodes" on page 849.

LocalEnvironment.SOAP
This subtree contains information that is stored by SOAPInput, SOAPAsyncResponse, or SOAPRequest nodes.

More details about the information that is stored in this subtree are in "WS-Addressing information in the local environment" on page 756.

If the message flow includes a SOAPAsyncResponse node, you can use this subtree to receive state and correlation information passed by a SOAPAsyncRequest node in another message flow.
More details about the information that is stored in this subtree are in "WS-Addressing with the SOAPAsyncRequest and SOAPAsyncResponse nodes" on page 754.

LocalEnvironment.ServiceRegistry
This subtree contains information for queries by the EndpointLookup and RegistryLookup nodes.

More details about the information that is stored in this subtree are in "Dynamically defining the search criteria" on page 791, "EndpointLookup node output" on page 793, and "RegistryLookup node output" on page 795.

LocalEnvironment.Wildcard
This subtree contains information about the wildcard characters that are stored by the FileInput node.

On the FileInput node you can specify a file name pattern that contains wildcard characters.

More details about the information that is stored in this subtree are in "Using local environment variables with file nodes" on page 849.

LocalEnvironment.Adapter
This subtree contains information that is stored by the WebSphere Adapters nodes.

For a WebSphere Adapters input node:

- **MethodName** is the name of the business method that corresponds to the Enterprise Information System (EIS) event that triggered this message delivery.

  The bindings for EIS events or business methods are created by the Adapter Connection wizard.
- **Type** describes the type of adapter that is being used:
  - SAP
  - Siebel
  - PeopleSoft

For a Websphere Adapters request node:

MethodName is the name of the business method that the request node must use.

LocalEnvironment.TCPIP

If the message flow includes a TCPIPClientReceive node or a TCPIPServerReceive node, you can override its TCPIP connection with elements in this subtree. See "TCPIPClientReceive node" on page 1234 and "TCPIPServerReceive node" on page 1264.

This subtree contains information that is stored by the TCPIPClientInput, TCPIPClientReceive, TCPIPServerInput, and TCPIPServerReceive nodes.

This information describes the connection that the node is using.

More details about the information that is stored in this subtree are in "TCPIPClientInput node" on page 1213, "TCPIPClientReceive node" on page 1234, "TCPIPServerInput node" on page 1245, and "TCPIPServerReceive node" on page 1264.

When the message flow processing is complete, the local environment tree is discarded.

The following samples demonstrate how to use the local environment to dynamically route messages based on the destination list:

- Airline Reservations
- Message Routing

The following sample uses the local environment tree to store information that is later added to the output message that is created by the message flow:

- User-defined Extension

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

Exception list tree structure:

The exception list tree is a part of the logical message tree in which the message flow writes information about exceptions that occur when a message is processed.

The root of the exception list tree is called ExceptionList, and the tree consists of a set of zero or more exception descriptions. The exception list tree is populated by the message flow if an exception occurs. If no exception conditions occur during message flow processing, the exception list that is associated with that message consists of a root element only. This list is, in effect, an empty list of exceptions.

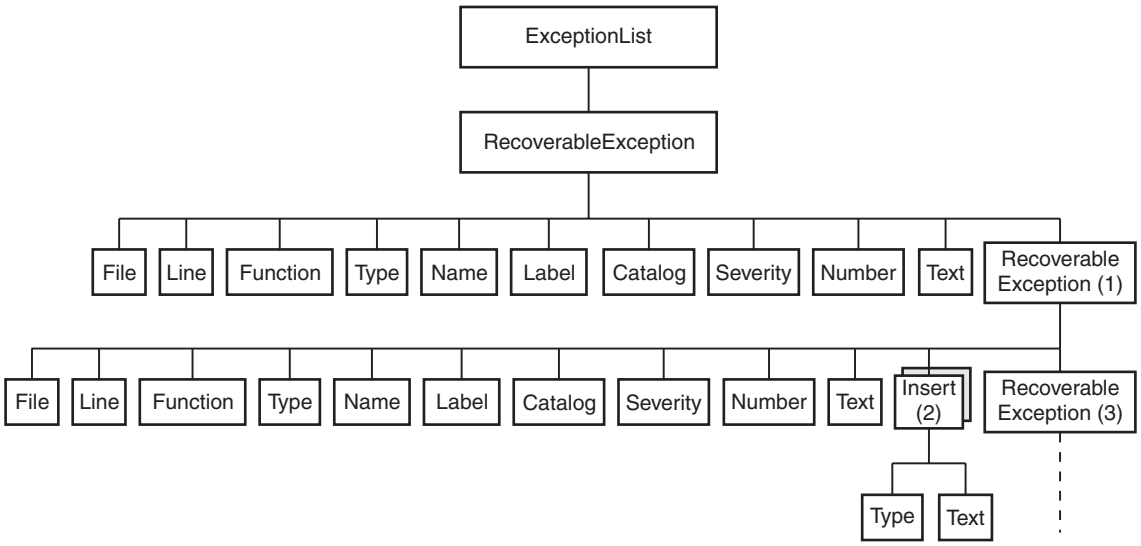
The exception list tree can be accessed by other nodes in the message flow that receive the message after the exception has occurred. You can modify the contents of the exception list tree only in a node that provides an interface to modify the outbound message tree; for example, the Compute node.

If an exception condition occurs, message processing is suspended and an exception is thrown. Control is passed back to a higher level; that is, an enclosing catch block. An exception list is built to describe the failure condition, and the whole message, together with the local environment tree, and the newly-populated exception list, is propagated through an exception-handling message flow path.

The child of ExceptionList is always RecoverableException. Typically, only one child of the root is created, although more than one might be generated in some circumstances. The child of ExceptionList contains a number of children, the last of which provides further information specific to the type of exception. The following list includes some of the exception types that you might see:

- FatalException
- RecoverableException
- ConfigurationException
- SecurityException
- ParserException
- ConversionException
- DatabaseException
- UserException
- CastException
- MessageException
- SqlException
- SocketException
- SocketTimeoutException
- UnknownException

The following figure shows the structure of the exception list tree for a recoverable exception:



The exception description structure can be both repeated and nested to produce an exception list tree. In this tree:

- The depth (that is, the number of parent-child steps from the root) represents increasingly detailed information for the same exception.
- The width of the tree represents the number of separate exception conditions that occurred before processing was abandoned. This number is usually one, and results in an exception list tree that consists of a number of exception descriptions that are connected as children of each other.
- At the numbered points in the tree:
  1. This child can be one of RecoverableException, ParserException, DatabaseException, UserException, ConversionException, or MessageException. All of these elements have the children shown; if present, the last child is the same element as its parent.

2. This element might be repeated.
3. If present, this child contains the same children as its parent.

The children in the tree take the form of a number of name-value elements that give details of the exception, and zero or more name elements whose name is Insert. The NLS (National Language Support) message number identified in a name-value element identifies a WebSphere Message Broker error message. The Insert values are used to replace the variables in this message and provide further detail about the cause of the exception.

The name-value elements in the exception list shown in the figure above are described in the following table.

Name		Type	Description
File[1]		String	C++ source file name
Line[1]		Integer	C++ source file line number
Function[1]		String	C++ source function name
Type[2]		String	Source object type
Name[2]		String	Source object name
Label[2]		String	Source object label
Text[1]		String	Additional text
Catalog[3]		String	NLS message catalog name[4]
Severity[3]		Integer	1 = information 2 = warning 3 = error
Number[3]		Integer	NLS message number[4]
Insert[3]	Type	Integer	The data type of the value: 0 = Unknown 1 = Boolean 2 = Integer 3 = Float 4 = Decimal 5 = Character 6 = Time 7 = GMT Time 8 = Date 9 = Timestamp 10 = GMT Timestamp 11 = Interval 12 = BLOB 13 = Bit Array 14 = Pointer
	Text	String	The data value

Notes:

1. Do not use the File, Line, Function, and Text elements for exception handling decision making. These elements ensure that information can be written to a log for use by IBM Service personnel, and are subject to change in both content and order.

2. The Type, Name, and Label elements define the object (usually a message flow node) that was processing the message when the exception condition occurred.
3. The Catalog, Severity, and Number elements define an NLS message: the Insert elements that contain the two name-value elements shown define the inserts into that NLS message.
4. NLS message catalog name and NLS message number refer to a translatable message catalog and message number.

When the message flow processing is complete, the exception list tree is discarded.

The following sample uses the exception list in the XML_Reservation message flow to pass error information to the Throw node, which generates an error message that includes the information from ExceptionList:

- Airline Reservations

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

Correlation names

A correlation name is a field reference that identifies a well-defined starting point in the logical message tree and is used in field references to describe a standard part of the tree format.

When you access data in any of the four trees (message, environment, local environment, or exception list), the correlation names that you can use depend on the node for which you create ESQL or mappings, and whether the node creates an output message. For example, a Trace node does not alter the content of the message as it passes through the node, but a Compute node can construct a new output message.

You can introduce new correlation names with SELECT expressions, quantified predicates, and FOR statements. You can create non-correlation names in a node by using reference variables.

Correlation names in nodes that do not create an output message: Most message flow nodes do not create an output message; all ESQL expressions that you write in ESQL modules or in mappings in these nodes refer to just the input message. Use the following correlation names in the ESQL modules that you write for Database and Filter nodes:

Root The root of the message passing through the node.

Body The last child of the root of the message; that is, the body of the message. This name is an alias for `Root.*[<]`.

For a description of how to use the asterisk (*) in field references, see "Using anonymous field references" on page 335.

Properties
The standard properties of the input message.

Environment
The structure that contains the current global environment variables that are available to the node. Environment can be read and updated from any node for which you can create ESQL code or mappings.

LocalEnvironment
The structure that contains the current local environment variables that are available to the node. LocalEnvironment can be read and updated from any node for which you can create ESQL code or mappings.

DestinationList
The structure that contains the current local environment variables available to the node. Its preferred name is LocalEnvironment, although the DestinationList correlation name can be used for compatibility with earlier versions.

ExceptionList
The structure that contains the current exception list to which the node has access.

You cannot use these correlation names in the expression of any mapping for a Mapping, Extract, Warehouse, DataInsert, DataUpdate, or DataDelete node.

Correlation names in nodes that create an output message: If you are coding ESQL for a Compute node, the correlation names must distinguish between the two message trees involved: the input message and the output message. The correlation names in ESQL in these nodes are:

InputBody
The last child of the root of the input message. This name is an alias for `InputRoot.*[<]`.

For a description of how to use *, see "Using anonymous field references" on page 335.

InputRoot
The root of the input message.

InputProperties
The standard properties of the input message.

Environment
The structure that contains the current global environment variables that are available to the node. Environment can be read and updated.

InputLocalEnvironment
The structure that contains the local environment variables for the message passing through the node.

InputDestinationList
The structure that contains the local environment variables for the message passing through the node. Use the correlation name InputDestinationList for compatibility with earlier versions; if compatibility is not required, use the preferred name InputLocalEnvironment

InputExceptionList
The structure that contains the exception list for the message passing through the node.

OutputRoot
The root of the output message.

In a Compute node, the correlation name OutputBody is not valid.

OutputLocalEnvironment
The structure that contains the local environment variables that are sent out from the node.

While this correlation name is always valid, it has meaning only when the Compute Mode property of the Compute node indicates that the Compute node is propagating the LocalEnvironment.

OutputDestinationList
The structure that contains the local environment variables that are sent out from the node. Use the correlation name OutputDestinationList for compatibility with earlier versions; if compatibility is not required, use the preferred name OutputLocalEnvironment

OutputExceptionList
The structure that contains the exception list that the node is generating.

While this correlation name is always valid, it has meaning only when the Compute Mode property of the Compute node indicates that the Compute node is propagating the ExceptionList.

Predefined and self-defining messages

Both predefined and self-defining messages are supported.

Each message that flows through a broker has a specific structure that is meaningful to the applications that send or receive that message.

You can use:

- Messages that you have modeled in the Broker Application Development perspective of the workbench; these messages are referred to as *predefined messages*. A model-driven parser requires predefined messages.
- Messages that can be parsed without a model; these are called *self-defining messages*.

Predefined messages: When you create a message in the workbench, you define the fields (*Elements*) in the message, along with special field types that you might need, and specific values (Value Constraints) to which the fields might be restricted.

Every message that you model in the workbench must be a member of a message set. You can group related messages together in a message set; for example, request and response messages for a bank account query can be defined in a single message set.

When you deploy a message set to a broker, the Configuration Manager sends the definition of that message set to the broker in a form appropriate to the parser that is used to parse and write the message. The broker can manage multiple message dictionaries simultaneously.

For information about the benefits of predefining messages, see Why model messages?

The following samples demonstrate how to model messages in XML, CWF and TDS formats:

- Video Rental
- Comma Separated Value (CSV)

The following samples provide message sets for industry-standard message formats:

- EDIFACT
- FIX

- SWIFT
- X12

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

Self-defining messages: You can create and route messages that are self-defining. The best example of a self-defining message is an XML document.

You can also model self-defining messages in the workbench. However, you must not deploy these message sets to the brokers that support those message flows. For further information about why you might want to model these messages, see Why model messages?.

Several samples in the Samples Gallery, including the following, use self-defining XML messages because they do not require a message set, so that fewer resources must be defined:

- Large Messaging
- Airline Reservations

The following sample demonstrates how you can transform a message from self-defining XML to a predefined binary format:

- Coordinated Request Reply

The following sample demonstrates how you can extract information from an XML message and transform it into BLOB format for storage in a database:

- Data Warehouse

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

Parsers

A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

A parser is called when the bit stream that represents an input message is converted to the internal form that can be handled by the broker; this invocation of the parser is known as *parsing*. The internal form, a logical tree structure, is described in "Logical tree structure" on page 76. It is described as a tree because messages are typically hierarchical in structure; a good example of this structure is XML. The way in which the parser interprets the bit stream is unique to that parser; therefore, the logical message tree that is created from the bit stream varies from parser to parser.

The parser that is called depends on the structure of a message, referred to as the *message template*. Message template information comprises the *message domain*, *message set*, *message type*, and *physical format* of the message. Together, these values identify the structure of the data that the message contains.

A parser is also called when a logical tree that represents an output message is converted into a bit stream; this action by the parser is known as *writing*. Typically,

an output message is generated by an output node at the end of the message flow. However, you can connect more nodes to an output node to continue processing of the message.

The message domain identifies the parser that is used to parse and write instances of the message. The remaining parts of the message template, message set, message type, and physical format, are optional, and are used by model-driven parsers such as the MRM parser.

The logical structure of the message typically maps to the business content of the message; for example, it contains a customer name, address, and account number. It is only when you send a message across a connection that the physical characteristics are important, and influence the construction of the bit stream.

The broker requires access to a parser for every message domain to which your input messages and output messages belong. In addition, the broker requires a parser for every identifiable message header that is included in the input or output message. Parsers are called when required by the message flow.

Body parsers

WebSphere Message Broker provides built-in support for messages in the following message domains by providing message body parsers:

- MRM ("MRM parser and domain" on page 120)
- XMLNSC, XMLNS, and XML ("XML parsers and domains" on page 100)
- SOAP ("SOAP parser and domain" on page 96)
- DataObject ("DataObject parser and domain" on page 122)
- JMSMap and JMSStream ("JMS parsers and domains" on page 122)
- MIME ("MIME parser and domain" on page 123)
- BLOB ("BLOB parser and domain" on page 129)
- IDOC ("IDOC parser and domain" on page 129)

See "Which body parser should you use?" on page 94 for a discussion about which message body parser to use under what circumstances.

You specify which message domain to use for your message at the place in the message flow where parsing or writing is initiated.

- To parse a message bit stream, typically you set the *Message Domain* property of the input node that receives the message. But, if you are initiating the parse operation in ESQL, use the DOMAIN clause of the CREATE statement.

  The message tree that is created is described in "Message tree structure" on page 77. Its exact form might change as it progresses through the message flow, depending on what the nodes are doing.

  The last child element of the Root element of the message tree takes the name of the body parser that created the tree. For example, if the *Message Domain* property was set to `MRM`, the last child element of Root is called MRM, which indicates that the message tree is owned by the MRM parser.
- To write a message, the broker calls the owning body parser to create the message bit stream from the message tree.

Some body parsers are *model-driven*, which means that they use predefined messages from a message set when parsing and writing. The MRM, SOAP,

DataObject, IDOC, and (optionally) XMLNSC parsers are model-driven parsers. To use these parsers, messages must be modeled in a message set and deployed to the broker from the Message Broker Toolkit.

Other body parsers are *programmatic*, which means that the messages that they parse and write are *self-defining* messages, and no message set is required. See "Predefined and self-defining messages" on page 90.

When you use a model-driven parser, you must also specify the *Message Set* and, optionally, the *Message Type* and *Message Format* so that the parser can locate the deployed message definition with which to guide the parsing or writing of the message.

To parse a message bit stream, typically you set the *Message Set*, *Message Type*, and *Message Format* properties of the input node that receives the message. Or, if you are initiating the parse operation in ESQL, you use the SETTYPE, and FORMAT clauses of the CREATE statement. This information is copied into the *Properties* folder of the message tree.

To write a message, the broker calls the owning body parser to create the message bit stream from the message tree. If the parser is a model-driven parser, it uses the *MessageSet*, *MessageType*, and *MessageFormat* fields in the Properties folder.

Whether *Message Type* or *Message Format* are needed depends on the message domain.

Even if the body parser is not model-driven, it is good practice to create and use a message set in the Message Broker Toolkit, because it simplifies the development of your message flow applications, even though the message set is not deployed in the broker runtime environment. See Why model messages? for information about the advantages of creating a message set.

Header parsers

WebSphere Message Broker also provides parsers for the following message headers, which your applications can include in input or output messages:

- WMQ MQMD ("The MQMD parser" on page 1525)
- WMQ MQMDE ("The MQMDE parser" on page 1526)
- WMQ MQCFH ("The MQCFH parser" on page 1522)
- WMQ MQCIH ("The MQCIH parser" on page 1522)
- WMQ MQDLH ("The MQDLH parser" on page 1524)
- WMQ MQIIH ("The MQIIH parser" on page 1524)
- WMQ MQRFH ("The MQRFH parser" on page 1527)
- WMQ MQRFH2 and MQRFH2C ("The MQRFH2 and MQRFH2C parsers" on page 1527)
- WMQ MQRMH ("The MQRMH parser" on page 1527)
- WMQ MQSAPH ("The MQSAPH parser" on page 1528)
- WMQ MQWIH ("The MQWIH parser" on page 1529)
- WMQ SMQ_BMH ("The SMQ_BMH parser" on page 1529)
- JMS header (Representation of messages across the JMS Transport)
- HTTP headers (HTTP headers)

All header parsers are programmatic and do not use a message set when parsing or writing.

User-defined parsers

To parse or write message body data or headers that the supplied parsers do not handle, you can create user-defined parsers that use the WebSphere Message Broker user-defined parser programming interface.

Tip: No parser is provided for messages, or parts of messages, in the WMQ format `MQFMT_IMS_VAR_STRING`. Data in this format is often preceded by an MQIIH header (format `MQFMT_IMS`). WebSphere Message Broker treats such data as a BLOB message. If you change the CodedCharSetId or the encoding of such a message in a message flow, the `MQFMT_IMS_VAR_STRING` data is not converted, and the message descriptor or preceding header does not correctly describe that part of the message. If you need the data in these messages to be converted, use the MRM domain and create a message set to model the message content, or provide a user-defined parser.

Which body parser should you use?

The characteristics of the messages that your applications exchange indicate which body parser you must use.

WebSphere Message Broker provides a range of message parsers. Each parser processes either message body data for messages in a particular message domain (for example, XML), or particular message headers (for example, the MQMD).

Review the messages that your applications send to the broker, and determine to which message domain the message body data belongs, using the following criteria as a guide.

If your application data uses SOAP-based Web services, including SOAP with Attachments (MIME) or MTOM

Use the SOAP domain. The SOAP domain has built-in support for WS-Addressing and WS-Security standards.

If your application data is in XML format other than SOAP

The domain that you use depends on the nature of the XML documents and the processing that you want to perform. See "Which XML parser should you use?" on page 95

If your application data comes from a C or COBOL application, or consists of fixed-format binary data

Use the MRM domain with a Custom Wire Format (CWF) physical format.

If your application data consists of formatted text, perhaps with field content that is identified by tags, or separated by specific delimiters, or both

Use the MRM domain with a Tagged/Delimited String (TDS) physical format.

If your application data is created using the JMS API

The domain that you use depends on the type of the JMS message. For a full description of JMS message processing, see JMS message as input.

If your application data is from a WebSphere Adapter such as the adapters for SAP, PeopleSoft, or Siebel

Use the DataObject domain.

If your application data is in SAP text IDoc format, such as those exported using the WebSphere MQ Link for R3

Use the MRM domain with a Tagged/Delimited String (TDS) physical format.

If your application data is in MIME format other than SOAP with Attachments (for example, RosettaNet)

Use the MIME domain. If the message is multipart MIME, you might need to parse specific parts of the message with other parsers. For example, you might use the XMLNSC parser to parse the XML content of a RosettaNet message.

If you do not know, or do not need to know, the content of your application data Use the BLOB domain.

Which XML parser should you use?:

If your messages are general purpose XML documents, you can use one of the dedicated XML domains (XMLNSC or XMLNS) to parse the message, or you can use the MRM domain to parse the message.

Note: Although SOAP XML can be parsed using any namespace-aware XML parser, use the dedicated SOAP domain to parse SOAP XML because the SOAP domain provides full support for SOAP with Attachments, and standards such as WS-Addressing and WS-Security.

Note: The XML domain is deprecated. Do not use it for developing new message flows. The XML domain still works with existing message flows.

Which XML parser you choose depends on the nature of your XML messages, and the transformation logic that you want to use. The differentiating features of each domain are:

- The XMLNSC parser has a new architecture that gives significant performance improvements over the XMLNS and XML parsers.
- The XMLNSC parser can be used with or without an XML Schema that is generated from a message set. Using a message set with the XMLNSC parser allows the parser to operate in validating mode which provides the following capabilities:
  - XML Schema 1.0 compliant validation when parsing and writing.
  - The XML Schema indicates the real data type of a field in the message instead of always treating the field as a character string.
  - Base64 binary data can be automatically decoded.
- The MRM parser must be used with a message dictionary that is generated from a message set. This message dictionary enables the MRM parser to provide the following capabilities: For example:
  - Validation against the dictionary when parsing and writing. Note that validation is not XML Schema 1.0 compliant.
  - The dictionary indicates the real data type of a field in the message instead of always treating the field as a character string.
  - Base64 binary data can be automatically decoded.
- The XMLNS parser is programmatic and does not use a model when parsing.

  This means that:
  - All data in an XML message is treated as character strings.
  - Validation is not possible when parsing and writing.

- The MRM parser uses information from the XML physical format of a message set to simplify the task of creating transformation logic:
  - Date and time information can be extracted from a data value using a specified format string.
  - When creating output messages, the MRM parser can automatically generate the XML declaration, and other XML constraints.
- The XMLNSC and XMLNS parsers do not use XML physical format information from a message set. Transformation logic must explicitly create all constructs in an output message.
- The MRM parser discards some parts of an XML message when parsing; for example, white space formatting, XML comments, XML processing instructions, and inline DTDs. If you use this parser, you cannot create these constructs when building an output message.
- The XMLNSC parser, by default, discards white space formatting, XML comments, XML processing instructions, and inline DTDs. However, options are provided to preserve all of these constructs, except inline DTDs. You can create them all, except inline DTDs, when constructing an output message.
- The XMLNS parser preserves all parts of an XML document, including white space formatting. You can create all XML constructs when constructing an output message.
- The XMLNSC and MRM parsers build compact message trees that use fewer syntax elements than the XMLNS parser for attributes and simple elements. This makes these parsers more suitable than the XMLNS parser for parsing very large XML messages.
- The XMLNS parser builds a message tree that conforms more closely than the XMLNSC or MRM parsers to the XML Data Model. You might want to use this parser if you are using certain XPath expressions to access the message tree, and the relative position of parent and child nodes is important, or if you are accessing text nodes directly.

Tip: If performance is critical, use the XMLNSC domain.

Tip: If you need to validate the content and values in XML messages, use the XMLNSC domain.

Tip: If you need to preserve formatting in XML messages on output, use the XMLNSC domain with the option to retain mixed content.

Tip: If you require message tree to conform as closely as possible to the XML data model, perhaps because you are using certain XPath expressions to access the message tree, use the XMLNS domain.

Tip: If you are taking non-XML data that has been parsed by the CWF or TDS formats of the MRM domain, and merely transforming the data to the equivalent XML, use the MRM domain. This can be achieved by adding an XML physical format to the message set with default values, and changing the Message Format in the Properties folder of the message tree.

SOAP parser and domain

You can use the SOAP parser to create a common WSDL-based logical tree format for working with Web services, independent of the physical bitstream format.

Use the SOAP parser in conjunction with the SOAP nodes in your message flow.

Messages in the SOAP domain are processed by the SOAP parser. The SOAP parser creates a common logical tree representation for all SOAP-based Web services and validates the message against a WSDL definition. If a runtime message is not allowed by the WSDL, an exception is thrown, otherwise the `portType` and `operation` names from the WSDL are saved in the logical tree.

The SOAP domain offers `WS-*` processing, together with a canonical tree shape that is independent of the wire format (XML or MIME).

The standards supported are:

- WSDL 1.1
- SOAP 1.1 and 1.2
- MIME 1.0
- Message Transmission Optimization Mechanism (MTOM) 1.0

A WSDL 1.1 definition must be deployed to describe the Web service messages that the SOAP domain needs to parse and write at run time. The SOAP parser is, therefore, always model-driven. The bitstream format for these runtime messages can be SOAP 1.1 or SOAP 1.2, optionally wrapped by MIME as an SwA (SOAP with Attachments) or MTOM message.

When a message set that supports the SOAP domain is added to a broker archive (BAR) file, XML Schemas are created automatically from the message definition files in the message set, and any WSDL files in the message set are added to the BAR file. The WSDL and XML Schema are deployed to the broker and used by the SOAP parser.

If you want the SOAP domain to parse your SOAP Web service, you must:

1. Create a new message set, or locate an existing message set.
2. Ensure that either the message set has its *Default message domain* project set to `SOAP`, or the *SOAP* check box (under *Supported message domains*) is selected, to indicate that the message set supports the SOAP domain.
3. Import your WSDL file to create a message definition file. The WSDL is also added to the message set. Message definition files for the SOAP envelope and the SOAP logical tree are also added to the message set automatically.
4. Add the message set to a broker archive (BAR) file, which generates the required XML Schema and WSDL in a file with extension .xsdzip, and deploy the BAR file to the broker.
5. If you associate your WSDL with a SOAP node in your message flow, the *Message Set* property is automatically set in the node. The *Message domain* property is always pre-selected as `SOAP`.

Tip: The SOAP parser invokes the XMLNSC parser to parse and validate the XML content of the SOAP Web service. See “XMLNSC parser” on page 104.

SOAP message details:

A SOAP message consists of an `<Envelope>`, which is the root element in every SOAP message, and this contains two child elements, an optional `<Header>` and a mandatory `<Body>`.

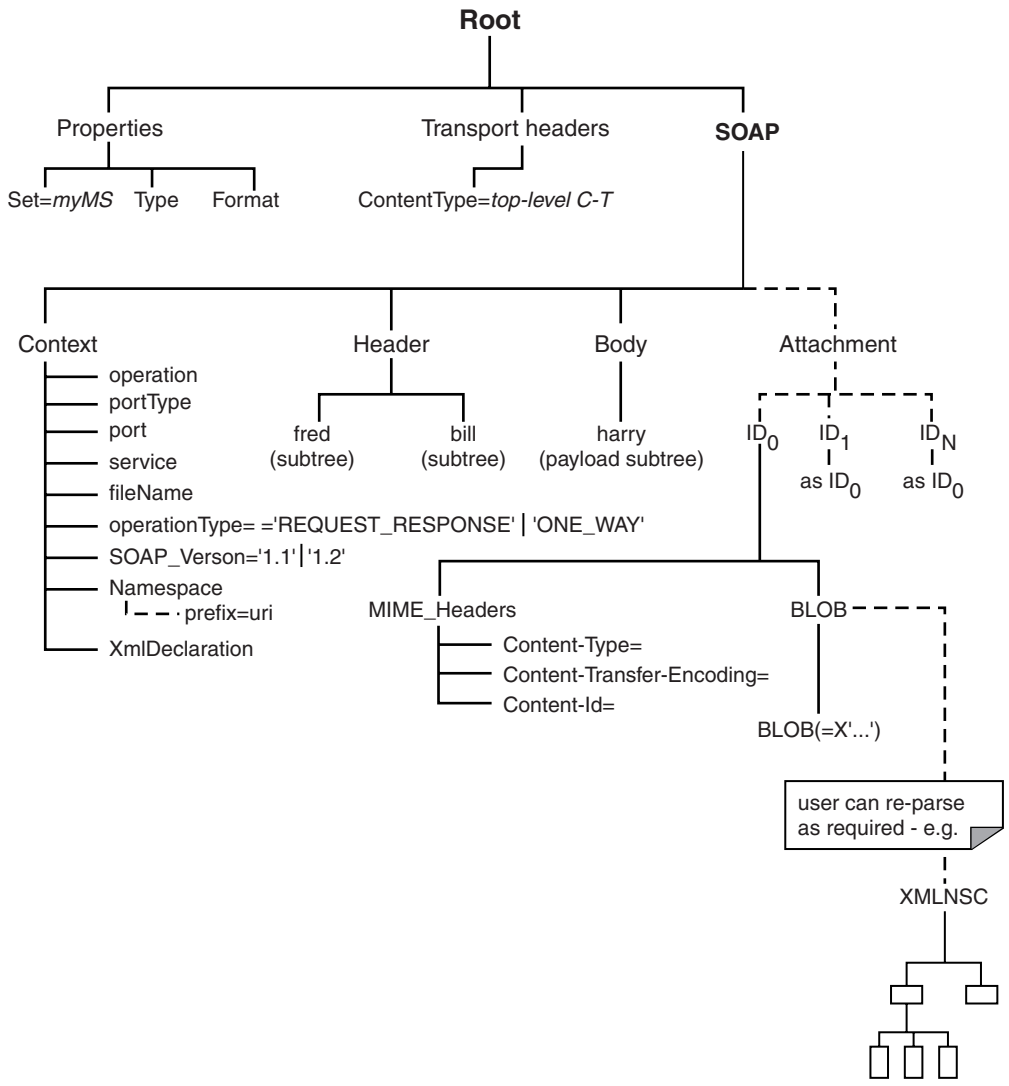
If the SOAP message has attachments, the 'envelope' is wrapped by MIME, or is encoded as MTOM.

For further information on the structure of a SOAP message, see "The structure of a SOAP message" on page 738.

SOAP tree overview:

This tree format allows you to access the key parts of the SOAP message in a convenient way.

This is a diagrammatic representation of the SOAP domain tree:



The SOAP tree contains the following elements:

SOAP.Header
Contains the SOAP header blocks (children of `Envelope.Header`)

SOAP.Body
Contains the SOAP payload (children of `Envelope.Body`)

The content of the `Body` subtree depends on the WSDL style.

SOAP.Attachment
Contains attachments for an `SwA` message in their non encoded format.

Note that attachments for an MTOM message are represented inline as part of the SOAP content in a base 64 representation.

SOAP.Context

Contains the following information:

- Input; populated by the SOAPInput node:
  - * operation - the WSDL operation name
  - * portType - the WSDL port type name
  - * port - the WSDL port name (if known)
  - * service - the WSDL service name (if known)
  - * fileName - the original WSDL file name
  - * operationType - one of 'REQUEST_RESPONSE', 'ONE_WAY', 'SOLICIT_RESPONSE', 'NOTIFICATION'
  - * SOAP_Version - one of '1.1' or '1.2'
  - * Namespace - Contains nameValue child elements; the name is the Namespace prefix, and the value is the Namespace URI as it appears in the bitstream.
  - * XmlDeclaration - represents the standard XML declaration.
- Output; the following fields can be placed in SOAP.Context to provide override information when SOAPRequest or SOAPAsyncRequest nodes serialize a SOAP message:
  - * SOAP_Version - one of '1.1' or '1.2'
  - * Namespace - Contains nameValue child elements that define the namespace prefix (the name) to be used for a specified namespace URI (the value).

    An output message uses the namespace prefixes defined here to qualify any elements in the corresponding namespaces.

    If the SOAP.Context was originally created at an input node, it might already contain all the namespace prefix definitions that you need.

    If SOAP.Context does not exist, or the outgoing message uses additional namespaces, the SOAP parser generates any required namespace prefixes automatically.

    Alternatively, you can specify your own namespace prefix; the specific name of a namespace prefix does not usually affect the meaning of a message, with one important exception. If the message content contains a qualified name, the message must contain a matching namespace prefix definition.

    For example, if the output message is a SOAP Fault containing a <faultcode> element with the value soapenv:Server, a namespace prefix (which is case sensitive) for soapenv must be defined in the logical tree:

```
-- Build SOAP Fault message. Note that as well as defining the correct
-- namespace for the Fault element, it is also necessary to bind the
-- namespace prefix used in the faultcode element (this is set up under
-- SOAP.Context.Namespace)

-- Send back a new user defined SOAP 1.2 fault message
DECLARE soapenv NAMESPACE 'http://www.w3.org/2003/05/soap-envelope';
DECLARE xml     NAMESPACE 'http://www.w3.org/XML/1998/namespace';
DECLARE myNS    NAMESPACE 'http://myNS';

SET OutputRoot.SOAP.Context.Namespace.(SOAP.NamespaceDecl)xmlns:soapenv = soapenv;
SET OutputRoot.SOAP.Context.Namespace.(SOAP.NamespaceDecl)xmlns:myNS = myNS;

SET OutputRoot.SOAP.Body.soapenv:Fault.soapenv:Code.soapenv:Value = 'soapenv:Receiver';
SET OutputRoot.SOAP.Body.soapenv:Fault.soapenv:Code.soapenv:Subcode.soapenv:Value = 'my:subcode value';
SET OutputRoot.SOAP.Body.soapenv:Fault.soapenv:Reason.soapenv:Text = 'my Reason string';
```

```
SET OutputRoot.SOAP.Body.soapenv:Fault.soapenv:Reason.soapenv:Text.(SOAP.Attribute)xml:lang = 'en';
SET OutputRoot.SOAP.Body.soapenv:Fault.soapenv:Node = 'my Node string';
SET OutputRoot.SOAP.Body.soapenv:Fault.soapenv:Role = 'my Role string';
SET OutputRoot.SOAP.Body.soapenv:Fault.soapenv:Detail.my:Text = 'my detail string';

-- Send back a new user defined SOAP 1.1 fault message
DECLARE soapenv NAMESPACE 'http://schemas.xmlsoap.org/soap/envelope/';
SET OutputRoot.SOAP.Context.Namespace.(SOAP.NamespaceDecl)xmlns:soapenv = soapenv;

SET OutputRoot.SOAP.Body.soapenv:Fault.faultcode = 'soapenv:Receiver';
SET OutputRoot.SOAP.Body.soapenv:Fault.faultstring = 'my fault string';
SET OutputRoot.SOAP.Body.soapenv:Fault.faultactor = 'my fault actor';
SET OutputRoot.SOAP.Body.soapenv:Fault.detail.Text = 'my detail string';
```

Only `Namespace`, `SOAP_Version`, and `XmlDeclaration` influence the bitstream generated for a SOAP tree; the other fields are for information only.

XML parsers and domains

You can use XML domains to parse and write messages that conform to the W3C XML standard.

The term *XML domains* refers to a group of three domains that are used by WebSphere Message Broker to parse XML documents.

When reading an XML message, the parser that is associated with the domain builds a message tree from the input bit stream. The input bit stream must be a well-formed XML document that conforms to the W3C XML Specification (version 1.0 or 1.1).

When writing a message, the parser creates an XML bit stream from a message tree.

The domains have different characteristics, for guidance about which domain to choose, see "Which XML parser should you use?" on page 95.

XMLNSC domain

The XMLNSC domain is the preferred domain for parsing all general purpose XML messages, including those messages that use XML namespaces. This parser is the preferred parser for the following reasons:

- The XMLNSC parser has an architecture that results in ultra-high performance when parsing all kinds of XML.
- The XMLNSC parser reduces the amount of memory that is used by the logical message tree that is created from the parsed message. The default behavior of the parser is to discard non-significant white space and mixed content, comments, processing instructions, and embedded DTDs; however controls are provided to retain mixed content, comments, and processing instructions, if required.
- The XMLNSC parser can operate as a model-driven parser, and can validate XML messages against XML Schemas generated from a message set, to ensure that your XML messages are correct.

XMLNS domain

If the XMLNSC domain does not meet your requirements, use the alternative namespace-aware domain and parser.

XML domain

The XML domain is not namespace-aware. It is deprecated and must not be used to develop new message flows.

The MRM domain also provides XML parsing and writing facilities. For guidance on when you might use MRM XML instead of one of the XML parsers, see “Which XML parser should you use?” on page 95.

By default, the three XML parsers are *programmatic* parsers and do not use a message set at run time when parsing and writing. However, the MLNSC parser can operate as a model-driven parser and can validate XML messages for correctness against XML Schemas generated from a message set.

When you use the XMLNS or XML parsers, or the XMLNSC parser without a message set, it is good practice to create and use a message set in the Message Broker Toolkit; this action simplifies the development of your message flow applications, even though the message set is not deployed to the broker run time.

For the advantages of creating a message set, see Why model messages?.

The XML parsers are on-demand parsers. For more information, see “Parsing on demand” on page 1449.

The topics in this information center provide a summary of XML terminology, concepts, and message constructs. These aspects are important when you use XML messages in your message flows.

Tip: For more detailed information about XML, see the World Wide Web Consortium (W3C) Web site.

Example XML message parsing: A simple XML message might take the following form:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE Envelope
PUBLIC "http://www.ibm.com/dtds" "example.dtd"
[<!ENTITY Example_ID "ST_TimeoutNodes Timeout Request Input Test Message">]
>
<Envelope version="1.0">
 <Header>
  <Example>&Example_ID;</Example>
  <!-- This is a comment  -->
 </Header>
 <Body  version="1.0">
  <Element01>Value01</Element01>
  <Element02/>
  <Element03>
   <Repeated>ValueA</Repeated>
   <Repeated>ValueB</Repeated>
  </Element03>
  <Element04><P>This is <B>bold</B> text</P></Element04>
 </Body>
</Envelope>
```

The following sections show the output that is created by the Trace node when this example message has been parsed in the XMLNS and XMLNSC parsers. They demonstrate the differences in the internal structures that are used to represent the data as it is processed by the broker.

Example XML Message parsed in the XMLNS domain: In the following example, the white space elements within the tree are present because of the space, tab, and line breaks that format the original XML document; for clarity, the actual characters in the trace have been replaced with `'WhiteSpace'`. White space within an XML element does have business meaning, and is represented by using the Content

syntax element. The XmlDecl, DTD, and comments, are represented in the XML domain using explicit syntax elements with specific field types.

```
(0x01000010):XMLNS        = (
    (0x05000018):XML      = (
      (0x06000011): = '1.0'
      (0x06000012): = 'UTF-8'
      (0x06000014): = 'no'
    )
    (0x06000002):         = 'WhiteSpace'
    (0x05000020):Envelope = (
      (0x06000004): = 'http://www.ibm.com/dtds'
      (0x06000008): = 'example.dtd'
      (0x05000021): = (
        (0x05000011):Example_ID = (
          (0x06000041): = 'ST_TimeoutNodes Timeout Request Input Test Message'
        )
      )
    )
    (0x06000002):         = 'WhiteSpace'
    (0x01000000):Envelope = (
      (0x03000000):version = '1.0'
      (0x02000000):        = 'WhiteSpace'
      (0x01000000):Header  = (
        (0x02000000):        = 'WhiteSpace'
        (0x01000000):Example = (
          (0x06000020): = 'Example_ID'
          (0x02000000): = 'ST_TimeoutNodes Timeout Request Input Test Message'
          (0x06000021): = 'Example_ID'
        )
        (0x02000000):        = 'WhiteSpace'
        (0x06000018):        = ' This is a comment  '
        (0x02000000):        = 'WhiteSpace'
      )
      (0x02000000):        = 'WhiteSpace'
      (0x01000000):Body    = (
        (0x03000000):version   = '1.0'
        (0x02000000):          = 'WhiteSpace'
        (0x01000000):Element01 = (
          (0x02000000): = 'Value01'
        )
        (0x02000000):          = 'WhiteSpace'
        (0x01000000):Element02 =
        (0x02000000):          = 'WhiteSpace'
        (0x01000000):Element03 = (
          (0x02000000):         = 'WhiteSpace'
          (0x01000000):Repeated = (
            (0x02000000): = 'ValueA'
          )
          (0x02000000):         = 'WhiteSpace'
          (0x01000000):Repeated = (
            (0x02000000): = 'ValueB'
          )
          (0x02000000):         = 'WhiteSpace'
        )
        (0x02000000):          = 'WhiteSpace'
        (0x01000000):Element04 = (
          (0x01000000):P = (
            (0x02000000):  = 'This is '
            (0x01000000):B = (
              (0x02000000): = 'bold'
            )
            (0x02000000):  = ' text'
          )
        )
```

```
      (0x02000000):          = 'WhiteSpace'
    )
    (0x02000000):        = 'WhiteSpace'
  )
```

Example XML Message parsed in the XMLNSC domain: The following trace shows the elements that are created to represent the same XML structure within the compact XMLNSC parser in its default mode. In this mode, the compact parser does not retain comments, processing instructions, or mixed text.

The example illustrates the significant saving in the number of syntax elements that are used to represent the same business content of the example XML message when using the compact parser.

By not retaining mixed text, all of the white space elements that have no business data content are no longer taking any space in the broker message tree at run time. However, the mixed text in `Element04.P` is also discarded, and only the value of the child folder, `Element04.P.B`, is held in the tree; the text `This is` and `text in P` is discarded. This type of XML structure is not typically associated with business data formats; therefore, use of the compact XMLNSC parser is typically desirable. However, if you want to this type of processing, either do not use the XMLNSC parser, or use it with *Retain mixed text mode* enabled.

The handling of the XML declaration is also different in the XMLNSC parser. The version, encoding, and stand-alone attributes are held as child entities of the XmlDeclaration, rather than as elements with a particular field type.

```
(0x01000000):XMLNSC     = (
    (0x01000400):XmlDeclaration = (
      (0x03000100):Version    = '1.0'
      (0x03000100):Encoding   = 'UTF-8'
      (0x03000100):StandAlone = 'no'
    )
    (0x01000000):Envelope       = (
      (0x03000100):version = '1.0'
      (0x01000000):Header  = (
        (0x03000000):Example = 'ST_TimeoutNodes Timeout Request Input Test Message'
      )
      (0x01000000):Body    = (
        (0x03000100):version   = '1.0'
        (0x03000000):Element01 = 'Value01'
        (0x01000000):Element02 =
        (0x01000000):Element03 = (
          (0x03000000):Repeated = 'ValueA'
          (0x03000000):Repeated = 'ValueB'
        )
        (0x01000000):Element04 = (
          (0x01000000):P = (
            (0x03000000):B = 'bold'
          )
        )
  )
```

The following samples use the XML parser to process messages:

- Coordinated Request Reply
- Large Messaging
- Message Routing

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

Some predefined message models are supplied with the Message Broker Toolkit and can be imported by using the New Message Definition File wizard and selecting the IBM supplied message option. See IBM supplied messages that you can import.

XMLNSC parser:

The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

The XMLNSC parser has a range of options that make it suitable for most XML processing requirements. Some of these options are only available in the XMLNSC parser.

Although the XMLNSC parser is capable of parsing XML documents without an XML Schema, extra features of the parser become available when it operates in model-driven mode. In model-driven mode, the XMLNSC parser is guided by an XML Schema, which describes the shape of the message tree (the logical model).

XML Schemas are created automatically from the content of a message set when the message set is added to a broker archive (BAR) file. The XML Schemas are deployed to the broker and used by the XMLNSC parser to validate your XML messages. Validation is fully compliant with the XML Schema 1.0 specification.

For guidance on when to use the XMLNSC domain and parser, see "Which XML parser should you use?" on page 95.

If you want the XMLNSC domain to parse a message, select *Message Domain* as `XMLNSC` on the appropriate node in the message flow. Additionally, if you want the XMLNSC parser to validate your messages, perform the additional steps that are described in "XMLNSC validation" on page 110.

Features of the XMLNSC parser

Feature	Present	Description
Namespace support	Yes	Namespace information is used if it is present. No user configuration is required. See "XML parsers namespace support" on page 118.
On-demand parsing	Yes	See "Parsing on demand" on page 1449.
Compact message tree	Yes	Less memory is used when building a message tree from an XML document. See "Manipulating messages in the XMLNSC domain" on page 408.
Opaque parsing	Yes	One or more elements can be parsed opaquely. See "XMLNSC opaque parsing" on page 109.

Feature	Present	Description
Ultra high performance	Yes	The architecture of the XMLNSC parser means that the parser's use of processor resources is significantly less than that of the other XML parsers.
Validation	Yes	See the table that follows this one.
Inline DTD support	Partial	Inline DTDs are processed but discarded. See “XMLNSC DTD support” on page 114.
XML Data Model compliance	Partial	The compact nature of the message tree means that some XPath queries are not supported.

The following features are only available when message validation is enabled. See “XMLNSC validation” on page 110.

Feature	Description
Message validation	Validates compliance with the XML Schema 1.0 specification.
xsi:nil support	Sets the value of an element to NULL if it has xsi:nil=”true” and the XML Schema indicates that it is nillable.
Default value support	Sets the value of an empty element, or missing attribute, to its default value, according to XML Schema rules.
Use correct simple types	Allows the use of the simple types that are defined in the XML Schema when building the message tree.
Base64 support	Converts base64 data to BLOB when parsing. Converts BLOB to base64 when writing.

If you specify the SOAP domain as the owner of a SOAP Web Services message, the SOAP parser invokes the XMLNSC parser in model-driven mode to parse the XML content of the SOAP message.

If you specify the DataObject domain as the owner of a WebSphere Adapter message, and the message is written to a destination other than a WebSphere Adapter, the DataObject parser invokes the XMLNSC parser to write the message as XML.

XMLNSC empty elements and null values:

Empty elements and null values occur frequently in XML documents.

A robust message flow must be able to recognize and handle empty elements and null values. Similarly, elements in a message tree might have a NULL value, an empty value, or no value at all. This topic explains the parsing and writing of

these values by the XMLNSC domain. For advice on good ESQL or Java coding practices, see “Handling null values” on page 130.

Parsing

Description	XML input parsed by XMLNSC	Value of ‘element' in message tree
Empty element value	<element/>	Empty string
Empty element value	<element></element>	Empty string
Folder with child elements	<element><childElement/ ></element>	No value
Nil element value	<element xsi:nil="true"/>	Empty string or NULL **Note:** Which value depends on whether the element definition is 'nillable' in the XML Schema

Note that both forms of an empty element result in the same value in the message tree.

Writing

Description	Value of ‘element' in message tree	XML output from XMLNSC parser
Empty element value	Empty string	<element/>
Null element value	NULL	<element/>
Folder with child elements	No value	<element><childElement/ ></element>

Empty elements

An empty element can take two forms in an XML document:

```
- <element/>
- <element></element>
```

The XMLNSC parser treats both forms in the same way. The element is added to the message tree with a value of “” (the empty string).

When a message tree is produced by the XMLNSC parser, it always uses the first form for elements that have a value of “” (the empty string).

Elements with an xsi:nil attribute

The following behavior is available only when validation is enabled in the message flow.

If an element in the input document has an xsi:nil attribute with the value ‘true', and the ‘nillable' property of the element definition in the XML schema is set to ‘true', the XMLNSC parser sets the value of the message tree element to NULL.

When a message tree is produced by the XMLNSC parser, if the value of the element is NULL and the element has no child elements, the element is written as <element/>; but, if the element has an xsi:nil attribute, it is written exactly like any other attribute.

Note that the XMLNSC parser produces only xsi:nil attributes that are already in the message tree. It does not automatically produce xsi:nil attributes for all message tree elements that have a NULL value and are 'nillable'.

XMLNSC: Using field types:

The XMLNSC parser sets the field type on every syntax element that it creates.

The field type indicates the type of XML construct that the element represents. The XMLNSC parser uses the field type when writing a message tree. The field type can be set by using ESQL or Java to control the output XML. The field types that are used by the XMLNSC parser must be referenced by using constants with names that are prefixed by 'XMLNSC.'

Tip: Field type constants that have the prefix 'XML.' are for use with the XMLNS and XML parsers only, and are not valid with the XMLNSC or MRM parsers.

Field types for creating syntax elements

Use the following field type constants to create syntax elements in the message tree. The XMLNSC parser uses these values when creating a message tree from an input message.

XML construct	XMLNSC Field Type constant	Value
Simple Element	XMLNSC.Field XMLNSC.CDataField	0x03000000 0x03000001
Attribute	XMLNSC.SingleAttribute XMLNSC.Attribute	0x03000101 0x03000100
Mixed content	XMLNSC.Value XMLNSC.CDataValue	0x02000000 0x02000001
Namespace Declaration	XMLNSC.SingleNamespaceDecl XMLNSC.NamespaceDecl	0x03000102 0x03000103
Complex element	XMLNSC.Folder	0x01000000
Inline DTD	XMLNSC.DocumentType	0x01000300
XML declaration	XMLNSC.XmlDeclaration	0x01000400
Entity reference	XMLNSC.EntityReference	0x02000100
Entity definition	XMLNSC.SingleEntityDefinition XMLNSC.EntityDefinition	0x03000301 0x03000300
Comment	XMLNSC.Comment	0x03000400
Processing Instruction	XMLNSC.ProcessingInstruction	0x03000401

Field types for path expressions (generic field types)

Use the following field type constants when querying the message tree by using a path expression; for example:

```
SET str = FIELDVALUE(InputRoot.e1.(XMLNSC.Attribute)attr1)
```

It is good practice to specify field types when querying a message tree built by the XMLNSC parser. This makes your ESQL code more specific and more readable, and it avoids incorrect results in some cases. However, care is required when choosing which field type constant to use. When you use the XMLNSC parser, use a generic field type constant when querying the message tree. This allows your path expression to tolerate variations in the input XML.

The generic field type constants are listed in the following table:

XML construct	XMLNSC Field Type constant	Purpose
Tag	XMLNSC.Element	Matches any tag, whether it contains child tags (XMLNSC.Folder) or a value (XMLNSC.Field)
Element	XMLNSC.Field	Matches a tag which contains normal text, CData, or a mixture of both. Does not match tags which contain child tags.
Attribute	XMLNSC.Attribute	Matches single-quoted and double-quoted attributes
Mixed content	XMLNSC.Value	Matches normal text, CData, or a mixture of both
XML Declaration	XMLNSC.NamespaceDecl	Matches single- and double-quoted declarations

If you write

```
InputRoot.e1.(XMLNSC.DoubleAttribute)attrName
```

your path expression does not match a single-quoted attribute. If you use the generic field type constant *XMLNSC.Attribute*, your message flow works with either single-quoted or double-quoted attributes.

Note that you should always use the field type constants and not their numeric values.

Field types for controlling output format

The following field types are provided for XML Schema and base64 support. Do not use these field type constants in path expressions; use them in conjunction with XMLNSC.Attribute and XMLNSC.Field to indicate the required output format for DATE and BLOB values. See "XMLNSC: XML Schema support" on page 418 for further information.

XMLNSC Field Type constant	Purpose	Value
XMLNSC.gYear	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gYear format.	0x00000010
XMLNSC.gYearMonth	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gYearMonth format.	0x00000040
XMLNSC.gMonth	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gMonth format.	0x00000020

XMLNSC Field Type constant	Purpose	Value
XMLNSC.gMonthDay	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gMonthDay format.	0x00000050
XMLNSC.gDay	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gDay format.	0x00000030
XMLNSC.base64Binary	The value must be a BLOB. The value is produced with base64 encoding.	0x00000060
XMLNSC.List	The element must be XMLNSC.Attribute or XMLNSC.Field. If the field type includes this value, the values of all child elements in the message tree are produced as a space-separated list.	0x00000070

Field types for direct output

Use the following field types to produce pre-constructed segments of an XML document. Character escaping is not done; therefore, take extra care not to construct a badly-formed output document. Use these constants only after carefully exploring alternative solutions.

XMLNSC Field Type constant	Purpose	Value
XMLNSC.Bitstream	The value of this syntax element must be a BLOB. The value is written directly to the output bit stream. For more information about its usage, see "Working with large XML messages" on page 404.	0x03000200
XMLNSC.AsisElementContent	The value of this syntax element must be CHARACTER. The value is written directly to the output bit stream. No character substitutions are performed. Use this element with care.	0x03000600

XMLNSC opaque parsing:

Opaque parsing is a performance feature that is offered by the XMLNSC domain.

If you are designing a message flow and you know that certain elements in a message are never referenced by the message flow, you can specify that these elements are parsed opaquely. This reduces the costs of parsing and writing the message, and might improve performance in other parts of the message flow.

Use the property *Opaque Elements* on the *Parser options* page of the relevant message flow node to specify the elements that you want to be parsed opaquely. This property specifies a list of element names. If an element in the input XML message is named in this list, the element is parsed as a single string.

An opaque element cannot be queried like an ordinary element; its value is the segment of the XML bit stream that belongs to the element, and it has no child elements in the message tree, even though it can represent a large subtree in the XML document.

When an opaque element is serialized, the value of the element is copied directly into the output bit stream. The string is converted to the correct code page, but no other changes are made. Because this might produce a bit stream that is not valid XML, some care is required.

Do not parse an element opaquely in any of the following cases:

- The message flow must access one of its child elements.
- The message flow changes the namespace prefix in a way that affects the opaque element or one of its child elements **and** the element is to be copied to the output bit stream.
- The element, or any child element, contains a reference to an internal entity that is defined in an inline DTD **and** the element is to be copied to the output bit stream.
- The element contains child attributes that have default values that are defined in an inline DTD **and** the element is to be copied to the output bit stream.

Make sure that you check the above points before you specify an element for opaque parsing.

Using opaque parsing has some drawbacks. When opaque parsing is enabled, some parts of the message are never parsed, and XML that is either badly formed or not valid is allowed to pass through the message flow without being detected. For this reason, if you enable validation, you cannot use opaque parsing.

The XMLNS domain offers a more limited opaque parsing facility, but this is provided only to support existing applications. Use the XMLNSC domain in new message flows if you want to use opaque parsing.

Specifying opaque elements for the XMLNSC parser:

Specify an element as an opaque element so that its content is ignored by the XMLNSC parser.

To specify the elements that are to be skipped by the XMLNSC parser:

1. Right-click the selected message flow node, click **Properties** and select **Parser Options**.
2. At the bottom of the XMLNSC Parser Options panel is an area that lists the elements that have already been selected as opaque elements. Click **Add** to add an element to this list. A new pane **Add Opaque elements Entry** opens.
3. In the **Add Opaque elements Entry** pane, specify the new XML element that you want to be opaquely parsed. Each opaque element must be specified as an ESQL element name or an XPath expression of the form `//prefix:name` (or `//name`, if your input document does not contain namespaces).

   Note: A prefix is used rather than a full URI to identify the namespace; for further information, see "XPath namespace support" on page 535.

Click **Edit** or **Delete** to edit the list of opaque elements.

XMLNSC validation:

The XMLNSC parser offers high-performance, standards-compliant XML Schema validation at any point in a message flow.

Validation of the input XML message or the message tree is performed against the XML Schemas that are deployed.

Validation is not the same as *parsing*. When parsing, the XMLNSC parser always checks that the input document is well-formed XML, according to the XML specification. If validation is enabled, the XMLNSC parser also checks that the XML document obeys the rules in the XML Schema.

Enabling XML Schema validation in a message flow

You must complete the following tasks to construct a message flow that validates an XML document in accordance with an XML Schema:

- Enable validation at the appropriate point in the message flow. This is typically achieved by setting the *Validate* property of the appropriate node to `Content` or `Content and Value`. See "Validating messages" on page 204.
- Ensure that all required XML Schema files are deployed. See "Deploying XML Schemas" later in this section.
- Specify the message set in which the XML Schemas are deployed. Typically, you specify the message set by selecting the Message Set property on the node.

Deploying XML Schemas

All XML Schemas that are used by WebSphere Message Broker must be created as message definition files within a message set.

To create and deploy a message set for XML Schema validation:

1. Create a new message set or locate an existing message set.
2. Ensure that the message set has its *Default message domain* set to `XMLNSC`, or that the XMLNSC check box under *Supported message domains* is selected, to indicate that the message set supports the XMLNSC domain.
3. Create a message definition file in the message set to represent your message. If you have an existing XML Schema or DTD that describes your message, you can import it. You can repeat this step for each message that you want to validate.
4. Add the message set to a broker archive (BAR) file, which generates the required XML Schema in a file with extension .xsdzip, and deploy the BAR file to the broker.

Standards compliant validation

XMLNSC validation complies fully with XML Schema v1.0 as defined in the specifications that are available at http://www.w3.org/TR/xmlschema-1/ and http://www.w3.org/TR/xmlschema-2/, with the following minor exceptions:

- Any floating point value that is smaller than 10E-43 is treated as zero.
- Any member of a group or complex type, that has both minOccurs > 1024 and maxOccurs > 1024, is validated as if minOccurs = 0 and maxOccurs is unbounded.

Validating XML v1.1 documents

You can validate documents that conform to the XML v1.1 specification, but support is limited by the fact that the XML Schema v1.0 documents must conform to XML v1.0.

As an example, you cannot always declare an XML v1.1 tag name in XML Schema v1.0. This limitation is not imposed by the XMLNSC parser implementation; it is a limitation of XML Schema v1.0.

Interpreting validation errors

A validation error is an error that results when the XML document breaks the rules that are defined in the XML schema. The XML Schema standard specifies exactly what these rules are, and how they should be applied. Validation errors that the XMLNSC parser issues contain information that links the error to the XML Schema rule that has been violated.

All validation errors are reported in BIP5025 or BIP5026 messages. Both messages begin with text in the following form:

```
XML schema validation error '[cvc-error key: error description]'
```

Examples:

```
'cvc-minInclusive-valid: The value "2" is not valid with respect to the minInclusive facet
with value "3" for type "po:itemCountType".'
```

```
'cvc-complex-type.2.4.a: Expecting element with local name "numItems" but saw "totalValue".'
```

To find the XML Schema rule that has been violated, open the XML Schema specification and search for the error key.

Example 1: Open http://www.w3.org/TR/xmlschema-1/ and search for ‘cvc-minInclusive-valid'. Follow the link to the XML Schema rules for the minInclusive facet.

Example 2: Open http://www.w3.org/TR/xmlschema-1/ and search for ‘cvc-complex-type'. Follow the link to the XML Schema rules for validating the content of a complex type. In this case, the error key contains extra information. The ‘2.4.a' refers to the exact sub-rule that was violated. It should not be included when searching for the rule.

If the XML Schema specification does not provide enough information, you can find more information using a search engine. The XML Schema standard is very widely used, and many online tutorials and other resources are available.

XMLNSC message tree options:

The XMLNSC options that are described in this section affect the parsing of an XML document by the XMLNSC parser. They have no effect on XML output.

Retain Mixed Content

Mixed content is XML text which occurs between elements.

```
<parent>
     <childElement1>Not mixed content</childElement1>
     This text is mixed content
     <childElement2>Not mixed content</childElement2>
</parent>
```

By default, the XMLNSC parser discards all mixed content. Mixed content is retained in the message tree if you select *Retain mixed content* in the Parser options page of the input node. For further information, see 'Handling mixed text' in “Manipulating messages in the XMLNSC domain” on page 408.

Retain Comments

By default, the XMLNSC parser discards all comments in the input XML. Comments are retained in the message tree if you select *Retain comments* in the Parser options page of the input node. For further information, see 'Handling comments' in “Manipulating messages in the XMLNSC domain” on page 408.

Retain Processing Instructions

By default, the XMLNSC parser discards all processing instructions in the input XML. Processing instructions are retained in the message tree if you select *Retain processing instructions* in the Parser options page of the input node. For further information, see 'Handling processing instructions' in “Manipulating messages in the XMLNSC domain” on page 408.

Build tree using XML Schema data types

By default, the XMLNSC parser uses the CHARACTER data type for all element and attribute values that the parser creates in the message tree. However, if you are using the XMLNSC parser to validate the XML document, you can select *Build tree using XML Schema data types* in the Parser options page of the input node. This causes element and attribute values to be cast to the message broker data type that most closely matches their XML Schema simple type. The exact mapping between XML schema types and message broker types can be found in “XMLNSC data types.”

XMLNSC data types:

The table shows the mapping between XML Schema simple types and the data types that the XMLNSC parser uses in the message tree when Build tree using XML Schema types is specified.

XML Schema type	Data type in message tree
anyURI	CHARACTER
base64Binary	BLOB
Boolean	BOOLEAN
Byte	INTEGER
Date	DATE
dateTime	TIMESTAMP
Decimal	DECIMAL
Double	FLOAT
duration	INTERVAL
ENTITIES	List of CHARACTER
ENTITY	STRING
Float	FLOAT
gDay	DATE
gMonth	DATE
gMonthDay	DATE
gYear	DATE
gYearMonth	DATE

XML Schema type	Data type in message tree
hexBinary	BLOB
ID	CHARACTER
IDREF	CHARACTER
IDREFS	List of CHARACTER
int	INTEGER
Integer	DECIMAL
language	CHARACTER
Long	INTEGER
Name	CHARACTER
NCName	CHARACTER
negativeInteger	DECIMAL
NMTOKEN	CHARACTER
NMTOKENS	List of CHARACTER
nonNegativeInteger	DECIMAL
nonPositiveInteger	DECIMAL
normalizedString	CHARACTER
NOTATION	CHARACTER
positiveInteger	DECIMAL
Qname	CHARACTER
short	INTEGER
string	CHARACTER
Time	DATETIME
Token	CHARACTER
unsignedByte	INTEGER
unsignedInt	INTEGER
unsignedLong	DECIMAL
unsignedShort	INTEGER

Note: Base64 decoding is automatically performed by the XMLNSC parser.

List types

In the message tree, a list type is represented as a parent node with an anonymous child node for each list item. This allows repeating lists to be handled without any loss of information.

If a list element repeats, the occurrences appear as siblings of one another, and each occurrence has its own set of child nodes representing its own list items.

XMLNSC DTD support:

The input XML message might contain an inline DTD.

Parsing

If the input XML document has an inline DTD, the XMLNSC parser reads and uses information in the DTD while parsing, but does not add the DTD information to the message tree.

Internal entity definitions in the DTD are used to automatically expand entity references that are encountered in the body of the document.

Attributes that are missing from the input document are automatically supplied with the default value specified in the DTD.

The XMLNSC parser never adds the DTD to the message tree because the information that it contains has already been used during the parse. This behavior keeps the message tree compact and reduces CPU usage, and means that the XMLNSC parser does not always produce exactly the same document as it parsed. However, the business meaning of the output document is not altered.

If these restrictions are a problem, the XMLNS domain and parser provide full support for parsing and writing of the DTD. See "XMLNS DTD support" on page 118.

Writing

The XMLNSC parser can produce a DTD that contains entity definitions only. This behavior allows the XMLNSC parser to be used for writing out XML documents that use internal entities (the most common reason for using a DTD). See "Manipulating messages in the XMLNSC domain" on page 408 for further details.

External DTDs

No support is offered for external DTDs

XMLNS parser:

The XMLNS parser is a flexible, general-purpose XML parser.

The XMLNS parser is not model-driven and does not use an XML Schema when parsing XML documents.

For guidance on when to use the XMLNS domain and parser, see "Which XML parser should you use?" on page 95.

If you want the XMLNS domain to parse a particular message, you must select *Message Domain* as `XMLNS` on the appropriate node in the message flow.

Features of the XMLNS parser

Feature	Present	Description
Namespace support	Yes	Namespace information is used if it is present. No user configuration is required. See "Namespace support" on page 478.
On-demand parsing	Yes	See "Parsing on demand" on page 1449.

Feature	Present	Description
Compact message tree	No	
Opaque parsing	Partial	Limited support from ESQL only for parsing a single element opaquely. See “XMLNS opaque parsing” on page 117.
Ultra high performance	No	
Validation	No	
Inline DTD support	Yes	Inline DTDs are processed and retained in the message tree. See “XMLNS DTD support” on page 118.
XML Data Model compliance	Yes	The resultant message tree conforms to the XML Data Model.

XMLNS empty elements and null values:

Empty elements and null values occur frequently in XML documents.

A robust message flow must be able to recognize and handle empty elements and null values. Similarly, elements in a message tree might have a NULL value, an empty value, or no value at all. This topic explains the parsing and writing of these values by the XMLNS domain. For advice on good ESQL or Java coding practices see “Handling null values” on page 130.

Parsing

Description	XML input parsed by XMLNS	Value of ‘element' in message tree
Empty element value	<element/>	Empty string
Empty element value	<element></element>	Empty string
Folder with child elements	<element><childElement/ ></element>	No value
Nil element value	<element xsi:nil="true"/>	Empty string

Note that both forms of an empty element result in the same value in the message tree.

Note also that a NULL value is never put into the message tree by the XMLNS parser.

Writing

Description	Value of ‘element' in message tree	XML output from XMLNS parser
Empty element value	Empty string	<element/>
Null element value	NULL	<element/>
Folder with child elements	No value	<element><childElement/ ></element>

Empty elements

An empty element can take two forms in an XML document:

```
- <element/>
- <element></element>
```

The XMLNS parser treats both forms in the same way. The element is added to the message tree with a value of "" (the empty string).

When a message tree is produced by the XMLNS parser, it always uses the first form for elements that have a value of "" (the empty string).

Elements with an xsi:nil attribute

The XMLNS parser treats the xsi:nil attribute exactly like any other attribute. When xsi:nil is encountered while parsing, it does not set the value of the parent element to NULL. If you require this behavior you should use the XMLNSC parser. When writing a message tree, if an xsi:nil attribute exists it will be produced in the same way as any other attribute.

XMLNS opaque parsing:

Opaque parsing is a performance feature that is offered by the XMLNS domain.

XMLNS opaque parsing has been superseded by the opaque parsing feature of the XMLNSC domain. Do not use the XMLNS parser for opaque parsing unless your message flow requires features that are only offered by the XMLNS parser.

If you are designing a message flow, and you know that a particular element in a message is never referenced by the message flow, you can specify that that element is to be parsed opaquely. This reduces the costs of parsing and writing the message, and might improve performance in other parts of the message flow.

To specify that an XML element is to be parsed opaquely, use an ESQL CREATE statement with a PARSE clause to parse the XML document. Set the FORMAT qualifier of the PARSE clause to the constant, case-sensitive string 'XMLNS_OPAQUE' and set the TYPE qualifier of the PARSE clause to the name of the XML element that is to be parsed in an opaque manner.

The TYPE clause can specify the element name with no namespace (to match any namespace), or with a namespace prefix or full namespace URI (to match a specific namespace).

XMLNS opaque elements cannot be specified via the node properties.

Consider the following example:

```
DECLARE soap NAMESPACE 'http://schemas.xmlsoap.org/soap/envelope/';

DECLARE BitStream BLOB ASBITSTREAM(InputRoot.XMLNS
                  ENCODING InputRoot.Properties.Encoding
                  CCSID InputRoot.Properties.CodedCharSetId);
--No Namespace
  CREATE LASTCHILD OF OutputRoot
    DOMAIN('XMLNS')
          PARSE (BitStream
```

```
                ENCODING InputRoot.Properties.Encoding
                CCSID InputRoot.Properties.CodedCharSetId
                FORMAT 'XMLNS_OPAQUE'
                TYPE 'Body');

--Namespace Prefix
  CREATE LASTCHILD OF OutputRoot
    DOMAIN('XMLNS')
          PARSE (BitStream
                ENCODING InputRoot.Properties.Encoding
                CCSID InputRoot.Properties.CodedCharSetId
                FORMAT 'XMLNS_OPAQUE'
                TYPE 'soap:Body');

--Namespace URI
  CREATE LASTCHILD OF OutputRoot
    DOMAIN('XMLNS')
          PARSE (BitStream
                ENCODING InputRoot.Properties.Encoding
                CCSID InputRoot.Properties.CodedCharSetId
                FORMAT 'XMLNS_OPAQUE'
                TYPE '{http://schemas.xmlsoap.org/soap/envelope/}:Body');
```

XMLNS DTD support:

The input XML might contain an inline DTD.

Parsing

If the input XML document has an inline DTD, the XMLNS parser reads and uses information in the DTD while parsing, and adds the DTD information to the message tree.

Internal entity definitions in the DTD are used to automatically expand entity references that are encountered in the body of the document.

Attributes that are missing from the input document are automatically supplied with the default value specified in the DTD.

Writing

The XMLNS parser can produce any inline DTD that has been constructed in the message tree.

External DTDs

No support is offered for external DTDs

XML parsers namespace support:

Namespaces in XML messages are supported by the XMLNSC and XMLNS parsers. Namespaces are not supported by the XML parser.

Parsing

The XMLNS and XMLNSC parsers can parse any well-formed XML document, whether or not the document contains namespaces. If elements or attributes have namespaces, those namespaces are applied to the elements and attributes in the message tree. Namespace prefix mappings are also carried in the message tree, and are used when serializing the message tree back to XML.

- If an element or attribute in the input XML has a namespace, the corresponding node in the message tree also has that namespace.
- If an element contains a namespace declaration (an xmlns attribute), a child element that contains its prefix and namespace URI is created in the message tree.

While the message is passing through a message flow, namespaces and namespace mappings can be modified using ESQL or any of the other transformation technologies that are offered by message broker.

Writing

Namespaces and their prefixes are preserved in the message tree when parsing, and are used when the XMLNS and XMLNSC parsers convert a message tree to an XML bitstream.

- When serializing a message tree, the parser scans for namespace declarations on each XML element. If any are found, it uses them to select the namespace prefixes in the output document.
- If an element in the message tree has a namespace, but there is no in-scope namespace declaration for its namespace URI, a valid namespace prefix is automatically generated and used in the output XML. Auto-generated prefixes have the form NS1, NS2, and so on.

Tip: If an element in the message tree has a child element that is a 'default namespace' declaration, every child of that element (whether an XML element or an XML attribute, at any nesting depth) must have a namespace. If this rule is not enforced message broker cannot generate correct XML output for the message tree.

XML parser:

The XML domain is very similar to the XMLNS domain, but the XML domain has no support for XML namespaces or opaque parsing.

The XML domain is deprecated, but existing message flows that use the XML domain continue to work. Use the XMLNSC domain when developing new message flows.

The XML parser is not model-driven and does not use an XML Schema when parsing XML documents.

If you want the XML domain to parse a particular message, you must select *Message Domain* as `XML` on the appropriate node in the message flow.

Tip: The XMLNSC and XMLNS parsers both support XML messages that do not use namespaces, with no extra configuration.

Features of the XML parser

Feature	Present	Description
Namespace support	No	
On-demand parsing	Yes	See "Parsing on demand" on page 1449.
Compact message tree	No	

Feature	Present	Description
Opaque parsing	No	
Ultra high performance	No	
Validation	No	
Inline DTD support	Yes	Inline DTDs are processed and retained in the message tree.
XML Data Model compliance	Yes	The resultant message tree conforms to the XML Data Model.

MRM parser and domain

You can use the MRM domain to parse and write a wide range of message formats.

The MRM domain can be used to parse and write a wide variety of message formats. It is primarily intended for non-XML message formats, but it can also parse and write XML. For guidance on when to consider using the MRM parser, instead of one of the XML parsers, to parse XML, see "Which XML parser should you use?" on page 95

The key features of the MRM domain are:

- Support for messages from applications that are written in C, COBOL, PL/I and other languages, by using the Custom Wire Format (CWF) physical format. This support includes the ability to create a message model directly from a C header file or COBOL copybook.
- Support for text messages, perhaps with field content that is identified by tags, separated by specific delimiters, or both, by using the Tagged Delimited String (TDS) physical format. This includes industry standards such as CSV, HL7, SWIFT, EDIFACT, and X12.
- Support for XML messages, including those that use XML namespaces, by using the XML physical format.

WebSphere Message Broker uses the MRM parser to read and write messages that belong to the MRM domain. When reading a message, the MRM parser constructs a message tree from a bit stream. When writing a message, the MRM parser creates a bit stream from a message tree. The MRM parser is always model-driven, and it is guided by a message dictionary that describes the shape of the message tree (the logical model) and the physical layout of the bytes or characters in the bit stream (the physical format). A message dictionary is created automatically from the content of a message set when it is added to the broker archive (BAR) file. Therefore, when you create a message set for use with the MRM domain, you must define both the logical model and the appropriate physical format information.

The operation of the parser depends on the physical format that you have associated with the input or output message:

- For a binary message, the parser reads a set sequence of bytes according to information in the CWF physical format, and translates them into the fields and values in the message tree.
- For a text message, the parser uses a key piece of TDS physical format information called *Data Element Separation* to decide how to parse each portion of the message bit stream. This informs the parser whether the message uses

delimiters, tags, fixed length elements, patterns, and so on. The parser then reads the data according to information in the TDS physical format, and translates it into the fields and values in the message tree.

- For an XML message, the parser reads the XML markup language (element tags and attributes), guided by information in the XML physical format, and translates them into the fields and values in the message tree.

Because the MRM parser is model-driven, it can perform validation of messages against the model that is defined in the deployed dictionary. The level of validation that is performed by the MRM parser is similar to that defined by XML Schema 1.0, but is not fully compliant. If you use XML messages, and you want fully compliant XML Schema 1.0 validation, use the XMLNSC domain.

The MRM parser is an on-demand parser. See "Parsing on demand" on page 1449.

If you want to use the MRM domain to parse a particular message:

1. Create a new message set with an appropriate CWF, TDS, or XML physical format; or locate an existing message set.
2. Ensure that the message set has its *Default message domain* set to `MRM`, or that the *MRM* check box under *Supported message domains* is selected to indicate that the message set supports the MRM domain.
3. Create a message definition file in the message set to represent your message, ensuring that both logical and physical format information is provided. If you have an existing C, COBOL, XML Schema, or DTD description of your message, you can import the description by using a wizard.
4. Add the message set to a broker archive (BAR) file which will generate a message dictionary for use by the MRM parser, and deploy the BAR file to the broker.
5. Select `MRM` as *Message Domain* on the appropriate node in your message flow.
6. Additionally set values for *Message Set*, *Message Type*, and *Message Format* on the node. *Message Type* is the name of the message in the message definition file.

The following samples all use the MRM parser to process messages:

- Video Rental
- Comma Separated Value (CSV)
- EDIFACT
- FIX
- SWIFT
- X12

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

Some predefined message models are supplied with the Message Broker Toolkit and can be imported by using the New Message Definition File From IBM supplied Message wizard. The CSV, ALE IDoc, and File IDoc models are specifically for use with the MRM domain. See IBM supplied messages that you can import.

IBM supplies predefined message sets for industry standard formats SWIFT, X12, EDIFACT, and FIX. Contact dubadapt@ie.ibm.com for more information.

DataObject parser and domain

Use the DataObject domain to parse and write messages for WebSphere Adapters.

Using the DataObject domain with WebSphere Adapters: You must use the DataObject domain when you use WebSphere Adapter nodes in your message flow.

WebSphere Message Broker uses the DataObject parser to read and write message from Enterprise Information Systems (EIS) such as SAP, PeopleSoft, and Siebel. Such messages belong to the DataObject domain.

When it receives a message from an adapter, the DataObject parser constructs a message tree from the business object that it receives from the EIS. When it writes a message, the DataObject parser creates from the message tree the business object that it sends to the EIS. The DataObject parser is always *model-driven*, and it is guided by the XML Schemas that model the EIS business objects. The XML Schemas are created automatically from the content of a message set when the message set is added to the broker archive (BAR) file.

If you want to parse a message using the DataObject domain, you must:

1. Create a message set, or locate an existing message set.
2. Ensure that either the message set has its *Default message domain* project set to `DataObject`, or the *DataObject* check box (under *Supported message domains*) is selected, to indicate that the message set supports the DataObject domain.
3. Create a message definition file in the message set to represent your EIS business object. Use the *New adapter connection* wizard to connect to the EIS and retrieve the Business object metadata.
4. Add the message set to a broker archive (BAR) file, which generates XML Schema for the DataObject parser to use, and deploy the BAR file to the broker.
5. If you associate your adapter inbound or outbound message with an adapter node in your message flow, the *Message Set* property is automatically set in the node. The *Message domain* property is always pre-selected as `DataObject`.

Tip: If a message that belongs to the DataObject domain is written to a destination other than a WebSphere Adapter, the DataObject parser invokes the XMLNSC parser to write the message as XML.

The following adapter samples use the DataObject parser to process messages:

- SAP Connectivity
- Twineball Example EIS Adapter

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

JMS parsers and domains

The JMSMap and JMSStream domains can be used for modeling messages that are produced by the implementations of the Java Messaging Service standard.

Use the JMSMap domain when handling JMS messages of type MapMessage. Use the JMSStream domain when handling JMS messages of type StreamMessage.

These message types appear in the broker in XML format, and are therefore supported in an identical way to XML domain messages.

For a full description of JMS MapMessage and StreamMessage processing, see WebSphere Broker JMS Transport.

MIME parser and domain

Use the MIME domain if your messages use the MIME standard for multipart messages.

The MIME (Multipurpose Internet Mail Extension) parser does not support the full MIME standard, but does support common uses of MIME. You can send the messages to the broker over HTTP or over other transport types, such as WebSphere MQ. Use the MIME domain if your messages use the MIME standard for multipart messages.

The MIME domain does not support Content-Type values with a media type of *message*.

To specify that a message uses the MIME domain, select `MIME` as the *Message Domain* on the relevant message flow node.

Use the MIME domain and parser to parse and write MIME messages. The MIME parser creates a logical tree, and sets up the broker ContentType property. You can use Compute nodes and JavaCompute nodes to manipulate the logical tree. Set the Content-Type value using the ContentType property in the MIME domain.

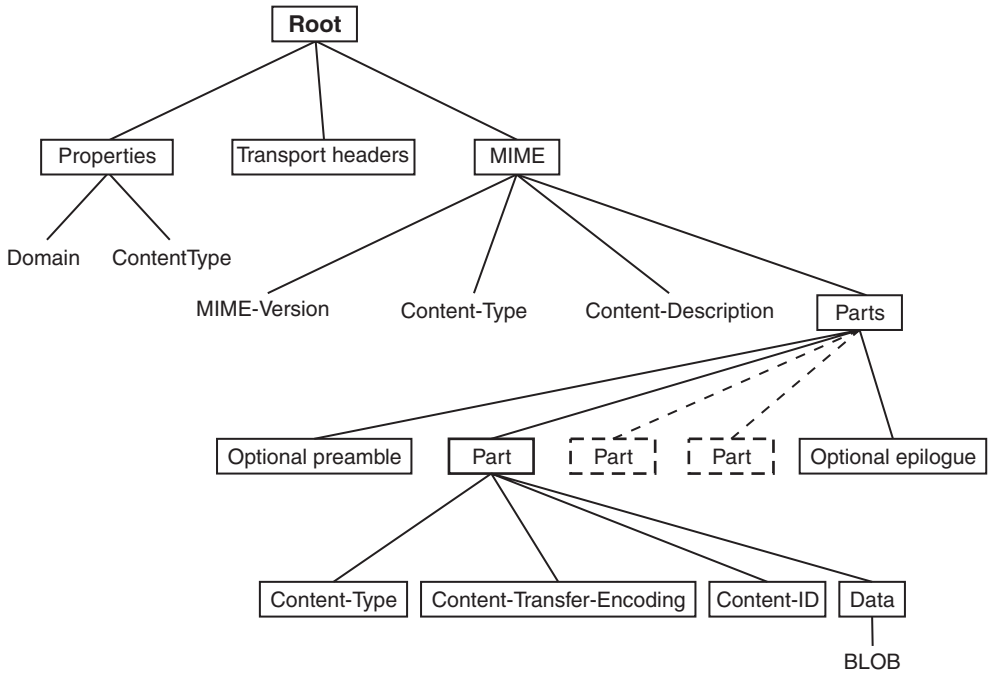
Example MIME message

The following example shows a simple multipart MIME message. The message shown is a SOAP with Attachments message with two parts: the root part and one attachment part. The boundary string *MIME_boundary* delimits the parts.

```
MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml
Content-Description: Optional description of message.

Optional preamble text
--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <rootpart@example.com>

<?xml version='1.0' ?>
<SOAP-ENV:Envelope
          xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

    <SOAP-ENV:Header xmlns:ins="http://myInsurers.com">
       <ins:ClaimReference>abc-123</ins:ClaimReference>
    </SOAP-ENV:Header>

    <SOAP-ENV:Body xmlns:ins="http://myInsurers.com">
       <ins:SendClaim>
          <ins:ClaimDetail>myClaimDetails</ins:ClaimDetail>
          <ins:ClaimPhoto>
            <href>cid:claimphoto@example.com</href>
          </ins:ClaimPhoto>
       </ins:SendClaim>
    </SOAP-ENV:Body>

</SOAP-ENV:Envelope>

--MIME_boundary
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary
Content-ID: <claimphoto@example.com>

myBinaryData
--MIME_boundary--
Optional epilogue text
```

Example MIME logical tree

The following diagram shows a MIME logical tree, which is described in "MIME tree details" on page 127. A MIME logical tree does not need to contain all of the children that are shown in the diagram. The value of the Content-Type header of a MIME message is the same as the ContentType field in the Properties subtree. The Transport-headers are headers from the transport that is used, such as an MQMD or HTTP.



You can further parse the BLOB data in the tree (for example, by using an ESQL CREATE statement) if you know about the format of that MIME part. You might be able to find information about the format from its Content-Type field in the logical tree. Alternatively, you might know the format that your MIME messages take, and be able to parse them appropriately. For example, you might know that the first MIME Part is always an XML message, and that the second MIME Part is a binary security signature.

You must specify how to parse other message formats, such as tagged delimited or binary data, within your message flow, because the MIME parser does not do this. You must also specify how to handle encoded and signed message parts, because the MIME parser does not process these.

Some pre-defined MIME message models are supplied with the workbench and can be imported using the New Message Definition From IBM Supplied Message wizard.

MIME messages:

A MIME message consists of both data and metadata. MIME metadata consists of HTTP-style headers and MIME boundary delimiters.

MIME headers

Each header is a colon-separated name-value pair on a line. The ASCII sequence <CR><LF> terminates the line. A sequence of these headers, called a header block, is terminated by a blank line: <CR><LF><CR><LF>. Any headers that are in this HTTP style can appear in a MIME document. Some common MIME headers are described in MIME standard header fields.

Content-Type

The only header that must be present is the *Content-Type* header. This header specifies the type of the data in the message. If the Content-Type value starts with "multipart", the message is a multipart MIME message. For multipart messages the Content-Type header must also include a boundary attribute that gives the text that is used to delimit the message parts. Each MIME part has its own Content-Type field that specifies the type of the data in the part. This can also be multipart, which allows multipart messages to be nested. MIME parts with any other Content-Type values are handled as BLOB data.

If a MIME document is sent over HTTP, the Content-Type header appears in the HTTP header block rather than in the MIME message body. For this reason, the broker manages the value of the Content-Type header as the ContentType property in the *Properties* folder of the logical tree. This allows the MIME parser to obtain the Content-Type value for a MIME document that is received over HTTP. If you need to either create a new MIME tree or modify the value of the Content-Type, set the Content-Type value using the ContentType property in the MIME domain. If you set the Content-Type value directly in the MIME tree or HTTP tree, this value might be ignored or used inconsistently. The following ESQL is an example of how to set the broker ContentType property:

```
SET OutputRoot.Properties.ContentType = 'text/plain';
```

Parsing

The MIME domain does not enforce the full MIME specification. Therefore, you can work with messages that might not be valid in other applications. For example, the MIME parser does not insist on a **MIME-Version** header. The MIME parser imposes the following constraints:

- The MIME headers must be properly formatted:
  - Each header is a colon-separated name-value pair, on a line of its own, terminated by the ASCII sequence <CR><LF>.
  - The header line must use 7-bit ASCII.
  - Semicolons are used to separate parameters:

    ```
    Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml
    ```

  - A header might contain a comment in parentheses, for example:

    ```
    MIME-Version: 1.0 (Generated by XYZ)
    ```

- A line that starts with white space is treated as a continuation of the previous line. Therefore, a long header can be split across more than one line.
- If two or more headers in a header block have the same name, their values are concatenated into a comma-separated list.
- A top-level MIME Content-Type header must be available. The header is not case-sensitive. If the transport is HTTP, any Content-Type value in the HTTP header is used as the top-level Content-Type. If the transport is not HTTP, the Content-Type must appear in the initial header block of the MIME message.
- The Content-Type value is a media type followed by the / character and a subtype. Examples of this are `text/xml` and `multipart/related`. The parser does not validate subtypes. The Content-Type value can be followed by one or more parameters that are separated by semicolons.
- If the media type of a message is multipart, a boundary attribute must provide the text that is used to delimit the separate MIME parts.
- Each individual MIME part can have its own Content-Type header. The part header can have a media type of multipart, so that multipart messages can be

nested. In this case, a valid boundary attribute must be provided, and its value must be different from any that has been previously defined in the message. MIME parts that have any other Content-Type value are handled as BLOB data.

- MIME multipart boundary delimiters are represented in 7-bit ASCII. The boundary delimiter consists of a line starting with a hyphen pair, followed by a boundary string. This sequence must not occur within the MIME message at any point other than as a boundary. A MIME end-delimiter is a hyphen pair, followed by the MIME boundary string, followed by a further hyphen pair. All delimiter lines must end in the ASCII sequence <CR><LF>. An example of a delimited message is:

```
--MIME_boundary
message data
--MIME_boundary
message data
--MIME_boundary--
```

  where *MIME_boundary* is the boundary delimiter string, and *message data* represents message data.
- The MIME media type *message* is not supported and results in an error at run time.
- Any preamble data (text between the initial MIME header block and the first boundary delimiter) or epilogue data (text after the final boundary delimiter) is stored in the logical tree as a value-only element. Preamble data and epilogue data can appear only as the first and last children, respectively, of a Parts node.
- The MIME parser does not support on-demand parsing and ignores the *Parse Timing* property. The parser does not validate MIME messages against a message model, and ignores the Message Broker Toolkit *Validate* property.

Special cases of multipart MIME

The MIME parser is intended primarily for use with multipart MIME messages. However, the parser also handles some special cases:

- Multipart MIME with just one part. The logical tree for the MIME part saves the Content-Type and other information as usual, but the Data element for the attachment is empty.
- Single-part MIME. For single-part MIME, the logical tree has no Parts child. The last child of the MIME tree is the Data element. The Data element is the parent of the BLOB that contains the message data.
- MIME parts with no content.

Secure MIME (S/MIME)

S/MIME is a standard for sending secure e-mail. S/MIME has an outer level Content-Type of *multipart/signed* with parameters **protocol** and **micalg** that define the algorithms that are used to encrypt the message. One or more MIME parts can have encoded content. These parts have Content-Type values such as *application/pkcs7-signature* and a Content-Transfer-Encoding of *base64*. The MIME domain does not attempt to interpret or verify whether the message is actually signed.

MIME tree details:

A MIME message is represented in the broker as a logical tree. When it writes a message, the MIME parser creates a message bit stream by using the logical message tree.

Logical tree elements

A MIME message is represented in the broker as a logical tree with the following elements:

- The root of the tree is a node called MIME.
- All correctly formatted headers are stored in the logical tree, regardless of whether they conform to the MIME standard. The headers appear in the logical tree as name=value, as shown here:

  ```
  Content-Type=text/xml
  ```
- A multipart MIME message is represented by a subtree with a root node called Parts.
- Any preamble or epilogue data associated with a multipart MIME message is represented by value-only elements appearing as the first and last children of Parts.
- In the special case of single-part MIME, the content is represented by a subtree with the root called Data.
- Each part of a multipart MIME message is represented by an element called Part with a child element for each MIME header, and a last child called Data.
- The Data element represents the content of a MIME part. This makes it easier to test for the presence of body content by using ESQL because the Data element is always the last child of its parent.

Writing MIME messages

When it writes a message, the MIME parser creates a message bit stream by using the logical message tree. The MIME domain does not enforce all of the constraints that the MIME specification requires, therefore it might generate MIME messages that do not conform to the MIME specification. The constraints that the MIME parser imposes are:

- The tree must have a root called MIME, and constituent Parts, Part, and Data elements, as described in "Logical tree elements."
- Exactly one Content-Type header must be present at the top level of the tree, or be available by using the ContentType property. Media subtypes are not validated.
- If the media type is *multipart*, a valid boundary parameter must also exist.
- Any constituent MIME parts can have exactly one Content-Type header. If the value of this header starts with *multipart*, it must also include a valid boundary parameter. The value of this boundary parameter must not be the same as other boundary parameter values in the definition.
- The MIME Content-Type value "message" is not supported and results in an error at run time.
- All name-value elements in the tree are written as `name: value` followed by the ASCII sequence `<CR><LF>`.

If you have other elements in the tree, the parser behaves in the same way as the HTTP header parser:

- A name-only element or a NameValue element with a NULL value results in `Name: NULL` .
- Any children of a name-value element are ignored.

The message flow must serialize subtrees if they exist; you can use the ESQL command ASBITSTREAM.

BLOB parser and domain

The BLOB message domain includes all the messages with content that cannot be interpreted and subdivided into smaller sections of information.

Messages in this domain are processed by the BLOB parser. The BLOB parser is a program that interprets a bit stream or message tree that represents a message that belongs to the BLOB domain. The parser then generates the corresponding tree from the bit stream on input, or a bit stream from the tree on output.

A BLOB message is handled as a single string of bytes, and although you can manipulate it, you cannot identify specific pieces of the byte string using a field reference, in the way that you can with messages in other domains.

You can process messages in the BLOB domain in the following ways:

- You can refer to the message content if you know the location (offset) of particular information within the message. You can specify offset values in ESQL statements within nodes in a message flow to manipulate the information.
- You can store the message in an external database, in whole or in part (where the part is identified by the offset of the data that is to be stored).
- You can use the Mapping node to map to and from a predefined BLOB message, and to map to and from items of BLOB data. The BLOB message cannot be:
  - The message content in a message where Content Validation is defined as Open or Open Defined (for example, the message body of a SOAP envelope)
  - The message represented by a wildcard inside another message

  The UnknownParserName field is ignored.

The BLOB message body parser does not create a tree structure in the same way that other message body parsers do. It has a root element BLOB, which has a child element, also called BLOB, which contains the data.

For example, `InputBody.BLOB.BLOB[10]` identifies the tenth byte of the message body; `substring(InputBody.BLOB.BLOB from 10 for 10)` references 10 bytes of the message data starting at offset 10.

If you want to use the BLOB parser to parse a particular message, select BLOB as the Message Domain on the relevant node in your message flow.

The following sample demonstrates how you can extract information from an XML message and transform it into BLOB format to store it in a database.

- Data Warehouse

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

IDOC parser and domain

The IDOC domain can be used to process messages that are sent to the broker by SAP R3 clients across the WebSphere MQ link for R3. Such messages are known as SAP ALE IDocs.

Note: The IDOC domain is deprecated and is not recommended for developing new message flows. Instead use the MRM domain with a TDS physical format. See "MRM parser and domain" on page 120.

A typical ALE IDoc message that has been sent from SAP to the WebSphere MQ link for R3 consists of an MQMD header, an MQSAPH header, and the ALE IDoc itself. The IDoc is made up of fixed size structures:

- The first structure is the Control Structure (DC). This is a complex element 524 bytes long that contains a fixed set of SAP-defined simple elements.
- One or more Data Structures (DDs). Each DD is a complex element 1063 bytes long that contains a fixed set of SAP-defined simple elements that occupies 63 bytes, followed by 1000 bytes of user-defined segment data.

WebSphere Message Broker uses the IDOC parser to read and write ALE IDocs that belong to the IDOC domain. When reading a message, the IDOC parser constructs a message tree from a bit stream. When writing a message, the IDOC parser creates a bit stream from a message tree.

The IDOC parser processes the SAP-defined elements in the DC, then, for each DD, the IDOC parser processes the SAP-defined elements, then calls the MRM parser to process the user-defined segment data, using its CWF physical format. The IDOC parser is therefore a model-driven parser, and requires that you create a message set in which to model the IDoc message, and deploy it to the broker.

If you want the IDOC domain to parse a particular message, you must:

1. Create a new message set with a CWF physical format, or locate an existing message set.
2. Ensure that either the message set has its *Default message domain* project set to `IDOC`, or the *IDOC* check box (under *Supported message domains*) is selected, to indicate that the message set supports the IDOC domain.
3. Create message definition files in the message set to represent your message. See Building the message model for the IDOC parser for the steps involved.
4. Add the message set to a broker archive (BAR) file which generates a message dictionary for use by the MRM parser, and deploy the BAR file to the broker.
5. Select *Message Domain* as `IDOC` on the appropriate node in your message flow.
6. Additionally, select *Message Set* and *Message Format* on the node. (You do not need to select *Message Type*).

Handling null values

A business message might contain fields that are either empty or have a specific out-of-range value. In these cases, the application that receives the message is expected to treat the field as if it did not have a value. The logical message tree supports this concept by allowing the value of any element to be set to NULL.

Ways to represent a null value: In an XML document, the usual way to represent a null value is to leave the element or attribute empty.

For example:`<price></price>` or `<element price=""/>`

The xsi:nil attribute provides a way to make this more explicit:`price=<xsi:nil="true"/>`

Some business messages use a special value to represent a null value:`<price>-999</price>`

This style of null representation is supported only by the MRM parser.

ESQL support for null values: Using ESQL, you can set the value of a message tree element to NULL:`SET OutputRoot.XMLNSC.myField VALUE = NULL;`

Note that this is quite different from `SET OutputRoot.XMLNSC.myField = NULL;` which would cause myField to be deleted from the message tree.

The same effect can be achieved using Java or a Mapping node.

XMLNSC parser support for null values: Typically, the XML parsers (XMLNSC, XMLNS, and XML) do not create null values in the message tree; an empty element or an empty attribute value merely produces an empty string value in the message tree.

If validation is enabled, the XMLNSC parser detects and processes any xsi:nil attributes in the input document. If the xsi:nil attribute is set to 'true', and the element is nillable, the attribute's parent element in the message tree is given a null value.

For more information about XML parser support for empty elements and null values, see “XMLNSC empty elements and null values” on page 105 and “XMLNS empty elements and null values” on page 116.

MRM parser support for null values: XML physical format

When parsing, the MRM XML parser can detect and process xsi:nil attributes in the input XML document. If the xsi:nil attribute is set to 'true', and the element is nillable, the attribute's parent element in the message tree is given a null value.

For information about enabling xsi:nil support in the MRM parser, see XML Null handling options.

The following topics provide more information about handling null values in the MRM parser:
- Custom wire format: NULL handling
- MRM XML physical format: NULL handling
- TDS format: NULL handling

All physical formats

The MRM parser can detect a null value that is represented by an out-of-range value. The null value must be specified in the physical format of the message set.

While parsing, the MRM parser checks the null value for each element in the message. If the value in the bit stream matches the null value in the message set, the MRM parser sets the value in the message tree to NULL.

The same check is performed when converting a message tree to a bit stream. If the value in the message tree is NULL, the MRM parser outputs the null value from the message set.

Properties

You can view and change properties that define broker characteristics, and those properties of associated resources such as message flows.

Broker properties

Each broker has a set of properties that define certain characteristics that you can control:

- You can set some broker properties only at the time that you create the broker by using the mqsicreatebroker command; for example, its associated queue manager.
- You can change some of the broker properties by using the mqsichangebroker command; for example, timeout values for processing configuration requests.
- You can view all broker properties by using the mqsireportbroker command.
- You can access some broker properties from ESQL and Java programs that you run in message flow nodes. For details of these options, see "Broker properties."

Node properties

Each built-in (supplied) node has at least one property that you can configure; some properties are mandatory, others are optional. In addition to setting these properties for each node when you add it to a message flow, you can also use the following configuration techniques:

- You can promote properties, so that you can set them at the message flow level. For information about how and why you might want to use this technique, see "Promoted properties" on page 133.
- You can configure properties when you add the message flow to a BAR file for deployment. In each node description, the subset of node properties that you can use in this way are identified. For more information, see Editing configurable properties.

Node properties are discussed in "Message flow nodes" on page 6.

User-defined properties

You can create your own properties when you create a message flow, and access those properties from ESQL and Java programs in your message flow nodes. For further details, see "User-defined properties" on page 134

Configurable service properties

You can modify some of the properties of the configurable services that are supplied to enhance your message flow processing options. For example, configurable services are supplied to communicate with enterprise information systems such as SAP.

You can also create your own configurable services, as variants of the supplied services, or to provide additional options.

For further information, see "Configurable services" on page 66.

Broker properties

For each broker, WebSphere Message Broker maintains a set of properties. You can access some of these properties from your ESQL programs. A subset of the properties is also accessible from Java programs. It can be useful, at run time, to have real-time access to details of a specific node, flow, or broker.

Broker properties are divided into four categories:

- Properties that relate to a specific node
- Properties that relate to nodes in general
- Properties that relate to a message flow
- Properties that relate to the execution group

For a description of the broker, flow, and node properties that are accessible from ESQL and Java, see Broker properties that are accessible from ESQL and Java.

Broker properties have the following characteristics.

- They are grouped by broker, execution group, flow, and node.
- They are case sensitive. Their names always start with an uppercase letter.
- They return NULL if they do not contain a value.

All nodes that allow user programs to edit ESQL support access to broker properties. These nodes are:

- Compute nodes
- Database nodes
- Filter nodes
- All derivatives of these nodes

User-defined properties can be queried, discovered, and set at run time to dynamically change the behavior of a message flow. You can use the Configuration Manager Proxy (CMP) API to manipulate these properties, which can be used by a systems monitoring tool to perform automated actions in response to situations that it detects in the monitored systems. For more information, see "User-defined properties" on page 134.

A *complex property* is a property to which you can assign multiple values. Complex properties are displayed in a table in the Properties view, where you can add, edit, and delete values, and change the order of the values in the table. You cannot promote complex properties; therefore, they do not appear in the Promote properties dialog box. Nor can you configure complex properties; therefore, they are not supported in the Broker Archive editor. For an example of a complex property, see the Query elements property of the DatabaseRoute node.

For more information about editing a node's properties, see "Configuring a message flow node" on page 276.

Promoted properties

A promoted property is a message flow node property that has been promoted to the level of the message flow in which it is included.

A message flow contains one or more message flow nodes. You can promote the properties of a message flow node to apply to the message flow to which it belongs. In this case, any user of the message flow can override these promoted properties at the message flow level, without being aware of the message flow's internal structure.

You can promote compatible properties (that is, properties that represent comparable values) from more than one node to the same promoted property; you can then set a single property that affects multiple nodes.

For example, you might want to set the name of a data source as a property of the message flow, rather than a property of each individual node in the message flow that references that data source. You create a message flow that accesses a database called SALESDATA. However, while you are testing the message flow, you want to use a test database called TESTDATA. If you set the data source properties of each individual node in the message flow to refer to SALESDATA, you can promote the data source property for each node in the flow that refers to it, and update the property to have the value TESTDATA; this value overrides the data source

properties on the nodes while you test the message flow (the promoted property always takes precedence over the settings for the properties in any relevant nodes).

A subset of message flow node properties is also configurable (that is, the properties can be updated at deployment time). You can promote configurable properties: if you do so, the promoted property (which can have a different name from the property or properties that it represents) is the one that is available to update at deployment time. Configurable properties are those associated with system resources; for example, queues and data sources. An administrator can set these properties at deployment time, without the need for a message flow developer.

You cannot promote a complex property, so it does not appear in the Promote properties dialog box. For more information about complex properties, see "Broker properties" on page 132.

User-defined properties

A user-defined property (UDP) is a property that is defined when you construct a message flow by using the Message Flow editor. This property can be used by the ESQL or Java program inside message flow nodes, such as a Compute node.

The advantage of UDPs is that their values can be changed by operational staff at deployment and run time. You do not need to change your application programs. For example, if you use the UDPs to hold data about your computer center, you can configure a message flow for a particular computer, task, or environment at deployment time, without having to change the code at the message node level.

When you launch the Message Flow editor to either create a message flow or modify an existing message flow, as well as deciding which nodes are required in the message flow, you also have the option (provided by the tab) of defining and giving initial values to some user-defined properties. Use the **User Defined Properties** tab at the bottom of the edit window. See Message Flow editor for more information.

As well as being defined using the Message flow editor, you must also define a UDP either by using a DECLARE statement with the EXTERNAL keyword in all ESQL programs that use it, or by calling the getUserDefinedAttribute method in all JavaCompute nodes that use it.

See the DECLARE statement for details of the DECLARE statement, and see "Accessing message flow user-defined properties from a JavaCompute node" on page 540 for more information about how to use a UDP in a JavaCompute node.

Values that you give to a UDP when you define it in a message flow override the value of that variable in your ESQL program.

You can also modify the value of a UDP at deployment time by using the Broker Archive editor to edit the BAR file. This value overrides the value that was set when you defined the message flow.

The value of the UDP is set at the message flow level, and is the same for all eligible nodes that are contained in the flow. An *eligible node* is a node that supports UDPs and is within the scope of the declaration that declares the UDP to your application. For example, if you use the Message Flow editor to change the value of a user property called timezone, which is declared in a schema called

mySchema, in a message flow called *myFlow*, the UDP is available at run time to all the nodes in *myFlow* that support UDPs and that fall within *mySchema*.

Similarly, if you use the Message Flow editor to change the value of a user-defined property in a subflow, the newly edited property is available to all the nodes in the subflow that support UDPs, and that are within the scope of the declaration. The property is not available, for example, to nodes in the parent flow.

A UDP has global scope and is not specific to a particular subflow. If you reuse a subflow in a message flow, and those subflows have identical UDPs, you cannot set the UDPs to different values.

Controlling user-defined properties at run time

User-defined properties can be queried, discovered, and set at run time to dynamically change the behavior of a message flow. You can use the Configuration Manager Proxy (CMP) to manipulate these properties, which can be used by a systems monitoring tool to perform automated actions in response to situations that it detects in the monitored systems.

For example, a message flow contains a Route node, which is used to differentiate between the classes of customer that are defined in the message. The Route node has a user-defined property called ProcessClasses, which is set with an initial value of All. When ProcessClasses is set to All, the node routes messages from all classes of customer to its first terminal for immediate processing.

When certain conditions are detected (for example, the monitoring system detects that the request load is causing the service level agreement to fall below its target), the Route node must be set to pass requests from only "Gold" class customers for immediate processing, while other customer requests are sent to another output terminal, which queues them for later batch processing. Therefore, the monitoring application sets ProcessClasses to Gold, so that the Route node routes the less critical messages to the second terminal.

To make it easier to know what a user-defined property does, and what values it can have, adopt a suitable naming convention. For example, a user-defined property named property01, with an initial value of valueA is not as useful as a property named RouteToAorB with an initial value of RouteA.

For more information, see Setting user-defined properties at run time in a CMP application.

Precedence of UDP value overriding

You can define a user-defined property in the following ways:
- In the ESQL code
- In the Message Flow editor
- Through a BAR file override
- By using the CMP API

You define a UDP in ESQL, in the Message Flow editor, or through a BAR file override before BAR file deployment. A BAR file override takes precedence over changes in the Message Flow editor, and changes in the Message Flow editor take precedence over changes in the ESQL code.

The BrokerProxy.deploy() call can take precedence over the BAR file override, or the BAR file override can take precedence over the BrokerProxy.deploy() call. In both cases, changes that are made are retained when the broker is restarted.

The precedence of the values for user-defined properties is shown in the following sequence:

1. The user-defined property ProcessClasses is set to All in a message flow BAR file. After deployment of the BAR file, the value of ProcessClasses is All.
2. The same user-defined property (ProcessClasses) is set to Gold by using the CMP API to issue the call setUserDefinedProperty("ProcessClasses", "Gold"). After successful completion of the BrokerProxy.deploy() call, the value of ProcessClasses is Gold.
3. The broker is shut down and restarted. The value of ProcessClasses is still Gold.
4. The original flow BAR file is redeployed. After deployment, the value of ProcessClasses is All.

Message flow transactions

A transaction describes a set of updates that are made by an application program, which must be managed together. The updates might be made to one or more systems. The updates made by the program are controlled by the environment in which the program executes, and either all are completed, or none. This property of a transaction is known as *consistency*: transactions might have other properties of *atomicity*, *isolation*, and *durability*.

WebSphere Message Broker supports transactions, and every piece of data processed by a message flow has an associated transaction. A message flow transaction is started by the broker when input data is received at an input node in the flow; it is committed when the flow has finished with that message, or rolled back if an error occurs.

If the flow contains more than one input node, one transaction is started for each input node when it receives input data. A transaction is started for every type of input node, including user-defined input nodes.

The nodes that you include in your message flow provide specific processing of a message, according to the defined function of each node. The processing that they do includes internal work, some of which you can influence by configuring the node properties. Some nodes perform additional tasks that might affect systems that are external to the message flow or the broker.

If an external system, such as a database, supports that concept of commit and rollback, and it can take part in a broker transaction, you can configure the node so that the work it does is included in the flow transaction. Depending on the node, you can also specify if the work done in an external system that supports transactions is committed immediately, or when the message flow transaction completes.

Many of the resources with which your message flows can interact are controlled by resource managers that can participate in coordinated transactions; for example, databases, WebSphere MQ messages and queues, and JMS messages. Other resource managers do not provide transactional support; for example, the HTTP protocol and file systems.

Commit or rollback

If the resource can participate in a transaction, you can configure it so that the work it does is committed or rolled back only when the message flow completes, or when the node completes. Databases and WebSphere MQ queues are examples of resources that you can use in this way. If the resource does not have transactional behavior, all the work that it does is committed immediately. For example, files and HTTP connections do not support transactions.

Updates that are made by a message flow are committed when the flow processes the input message successfully. The updates are rolled back if both of the following conditions are met:

- A node in the flow throws an exception that is not caught by a node other than the input node (for example, the node itself, or a TryCatch node)
- The Catch terminal of the input node is not connected

On distributed systems, message flow transactions are managed by the broker, by default. These transactions are known as *broker-coordinated transactions* or *partially coordinated transactions*. When control returns to the input node when the flow finishes processing, the node either commits or rolls back the operations that have been taken, excluding the individual nodes that have been configured to perform their own commits and rollbacks, or that have no support for this option.

If more than one resource is accessed by the message flow, an error might occur that prevents all the resources committing all the work that has been done. The broker raises an exception, and handles exception processing in a way that is determined by the transport that is involved. For example, messages that were read from WebSphere MQ queues are restored to those queues, and fault messages are sent to applications that submitted a message across HTTP (because HTTP has no concept of rollback). Because of these actions, the status of the resources might become inconsistent.

If it is important that your data and operations remain consistent, and that all operations are committed, or rolled back if one or more operations fail, you can coordinate the activity of the message flow. Coordination is provided by an external transaction manager which uses XA protocols to interact with resource managers. The transaction manager is called by the input node when the message flow has concluded (successfully or with errors). The transaction manager, rather than the input node and the broker, interacts with the relevant resource managers to initiate the correct actions for each resource. Transactions that are controlled by a transaction manager in this way are known as *globally coordinated transactions*.

On z/OS, transactions are always globally coordinated; you do not have to choose or configure for this option.

For a more detailed description of the transaction model in the broker, see The transactional model.

The role of the transaction manager

The result of the actions taken by message flows is the same for both partially and globally coordinated transactions if the message flow is successful in all its actions. The advantage of a globally coordinated transaction is the ability to ensure that either all actions are committed, or none.

The external transaction manager, which operates a two-phase commit strategy, supports cases where one or more external resource managers are temporarily unavailable during commit processing. This potentially small window for failure might be costly for your business environment; the external transaction manager helps to eliminate the occurrence of a failure window. Therefore, the decision to include an external transaction manager, which involves a performance overhead, is an administrative decision, not one to be taken at message flow design time.

An external transaction manager does not prevent message loss; even if you use transaction coordination, you must configure and code your message flows to handle potential errors as much as you can.

To configure a message flow to be globally coordinated, you must also set up your environment so that your resource managers are defined to the supported transaction manager:

- On distributed systems, transactions can be coordinated by WebSphere MQ
- On z/OS, all transactions are globally coordinated by Resource Recovery Service (RRS)

This configuration might require you to change settings in the transaction manager as well as the participating resource managers.

Database access modes and locks

You must use separate ODBC connections if you want to include nodes with Automatic transaction status and nodes with Commit transaction status in the same message flow, where the nodes operate on the same external database. Set up one connection for the nodes that are not to commit until the completion of the message flow, and a second connection for the nodes that are to commit immediately.

- If nodes with Commit transaction status are followed by a node with Automatic transaction status, the nodes with Commit transaction status commit independently of the flow transaction, and the nodes with Automatic transaction status commit at the end of the flow.
- However, if nodes with Automatic transaction status are followed by a node with Commit transaction status, and you do not use separate ODBC connections, error message BIP4001 is issued, because otherwise the node with Commit transaction status commits the work of the Automatic nodes prematurely.

Linux UNIX Windows On systems other than z/OS, individual relational databases might not support this mode of operation.

If you define more than one ODBC connection to the same data source, you might get database locking problems. In particular, if a node with Automatic transaction status carries out an operation, such as an INSERT or an UPDATE, that causes a database object (such as a table) to be locked, and a subsequent node tries to access that database object by using a different ODBC connection, an infinite lock (deadlock) occurs.

The second node waits for the lock acquired by the first to be released, but the first node does commit its operations and release its lock until the message flow completes. However, the flow cannot complete because the second node is waiting for the database lock held by the first node to be released.

Such a situation cannot be detected by a DBMS automatic deadlock-avoidance routine because the two operations are interfering with each other indirectly by using the broker.

You can use either of two options to avoid this type of locking problem:

- Design your message flow so that uncommitted (automatic) operations do not lock database objects that subsequent operations access across a different ODBC connection.
- Configure the lock timeout parameter of your database so that an attempt to acquire a lock fails after a specified length of time. If a database operation fails because of a lock timeout, an exception is thrown that the broker handles in the typical way.

For information concerning which database objects are locked by particular operations, and how to configure the lock timeout parameter of your database, consult your database product documentation.

Example of transactions in message flows

The following sample demonstrates the use of globally coordinated transactions, and the differences in the message flow when database updates are coordinated (the main flow), and when they are not (the error flow).

- Error Handler

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

Broker schemas

A broker schema is a symbol space that defines the scope of uniqueness of the names of resources defined within it. The resources include message flows and other optional resources such as ESQL files, Java files, and mapping files.

The broker schema is defined as the relative path from the project source directory to the flow name. When you first create a message flow project, a default broker schema named `(default)` is created within the project.

You can create new broker schemas to provide separate symbol spaces within the same message flow project. A broker schema is implemented as a folder, or subdirectory, within the project, and provides organization within that project. You can also use project references to spread the scope of a single broker schema across multiple projects to create an application symbol space that provides a scope for all resources associated with an application suite.

A broker schema name must be a character string that starts with a Unicode character followed by zero or more Unicode characters or digits, and the underscore. You can use the period to provide a structure to the name, for example `Stock.Common`. A directory is created in the project directory to represent the schema, and if the schema is structured using periods, further subdirectories are defined. For example, the broker schema `Stock.Common` results in a directory `Common` within a directory `Stock` within the message flow project directory.

If you create a resource (for example, a message flow) in the default broker schema within a project, the file or files associated with that resource are created in the directory that represents the project. If you create a resource in another broker schema, the files are created within the schema directory.

For example, if you create a message flow Update in the default schema in the message flow project Project1, its associated files are stored in the Project1 directory. If you create another message flow in the `Stock.Common` broker schema within the project Project1, its associated files are created in the directory `Project1\Stock\Common`.

Because each broker schema represents a unique name scope, you can create two message flows that share the same name within two broker schemas. The broker schemas ensure that these two message flows are recognized as separate resources. The two message flows, despite having the same name, are considered unique.

If you move a message flow from one project to another, you can continue to use the message flow within the original project if you preserve the broker schema. If you do this, you must update the list of dependent projects for the original project by adding the target project. If, however, you do not preserve the broker schema, the flow becomes a different flow because the schema name is part of the fully qualified message flow name, and it is no longer recognized by other projects. This action results in broken links that you must manually correct. For further information about correcting errors after moving a message flow, see "Moving a message flow" on page 263.

Do not move resources by moving their associated files in the file system; you must use the workbench to move resources to ensure that all references are corrected to reflect the new organization.

The following scope and reuse conditions apply when you create functions, procedures, and constants in a broker schema:

Functions

- Functions are locally reusable and can be called by module-scope subroutines or mappings within the same schema.
- Functions are globally reusable and can be called by other functions or procedures in ESQL or mapping files within any schema defined in the same or another project.

Procedures

- Procedures are locally reusable and can be called from module-scope subroutines in ESQL files within the same schema.
- Procedures are globally reusable and can be called by other functions or procedures in ESQL files within any schema defined in the same or another project.

Procedures cannot be used in mapping files.

Constants

- Constants are locally reusable and can be used where they are defined in any ESQL or mapping file within the same broker schema.
- Constants are not globally reusable; you cannot use a constant that is declared in another schema.

If you want to reuse functions or procedures globally:

- Specify the path of the function or procedure:
  - If you want to reuse a function or procedure in an ESQL file, either provide a fully-qualified reference, or include a PATH statement that defines the path.

    If you define the path, code the PATH statement in the same ESQL file as that in which the function is coded, but not within any MODULE.

– If you want to reuse a function in a mapping file, do one of the following:
  - Qualify the function in the Composition Expression editor.
  - Select **Organize Schema References** in the outline view. This detects dependent PATHs and automatically updates the reference.
  - Select **Modify Schema References** in the outline view. You can then select the schema in which the function is defined.

  (You cannot reuse a procedure in a mapping file.)
- Set up references between the projects in which the functions and procedures are defined and used.

Business-level monitoring

You can configure your message flow to emit event messages that can be used to support transaction monitoring and auditing, and business process monitoring.

Before you start:

An event is a message that a message flow publishes when something interesting happens. The message contains information about the source of the event, the time of the event, and the reason for the event. The event can include the message bit stream, and can also include selected elements from the message body. These fields can be used to correlate messages that belong to the same transaction, or to convey business data to a monitoring application.

Before starting to configure monitoring, read the following information.

1. Understand the basic concepts; see "Monitoring basics."
2. Decide what type of monitoring you intend to do; see "Monitoring scenarios" on page 143.
3. Decide whether to use monitoring properties, or a monitoring profile; see "Deciding how to configure monitoring events for message flows" on page 144.

To receive monitoring events, you must take the following steps:

1. Configure event sources on the flow, either by monitoring properties or a monitoring profile.
2. Ensure that the event sources are enabled; use the mqsireportflowmonitoring command to report settings. Use the monitoring properties for the node, or the mqsichangeflowmonitoring `-i` command to enable event sources, if appropriate.
3. Activate monitoring for the flow; use the mqsichangeflowmonitoring `-c` command.
4. Subscribe to the topic for the flow:
   `$SYS/Broker/`*`brokerName`*`/Monitoring/`*`executionGroupName`*`/`*`flowName`*

Monitoring basics

Message flows can be configured to emit events. The events can be read and used by other applications for transaction monitoring, transaction auditing, and business process monitoring.

Monitoring Events

A monitoring event is an XML document that conforms to the monitoring event schema. Each event contains the following information:

- Source of the event

- Name of the event
- Creation time
- Correlation ID for events emitted by the same transaction or unit of work

Additionally, a monitoring event can contain the following items:
- Application data extracted from the message
- Part or all of the message bit stream

See “The monitoring event” on page 1495 for more details

Event Sources

A message flow can emit two kinds of events:

Transaction events
Transaction events are emitted only from input nodes.

Terminal events
Terminal events are emitted from any terminal of any node, including input nodes.

An individual message flow can choose to emit transaction events, terminal events, or both kinds of event. You can configure, enable, and disable, both types of events in either of the following ways:
- Using the monitoring properties of the message flow.
- Using a monitoring profile configurable service.

The use of a monitoring profile configurable service overrides the monitoring properties of a message flow.

An event source address identifies an event source in a message flow.

Because terminal events can be emitted from any node in a message flow, they can be used as an alternative to dedicated event-emitting nodes or subflows such as that supplied in SupportPac IA9V.

Event sources emit events only if monitoring is activated for the message flow.

Terminal events

Any terminal in a message flow can be an event source. If the event source is active, it emits an event each time a message passes through the terminal.

Transaction events

Each input node in a message flow contains three events sources, in addition to terminal events.

Event source	Description
transaction.Start	The event is emitted when the message is read from the transport.
transaction.End	The event is emitted when WebSphere Message Broker has completed all processing of the message.

Event source	Description
transaction.Rollback	The event is emitted instead of transactionEnd if the message flow throws an exception which is not caught and processed within the message flow.

If a message flow handles its own exceptions, a transaction end event, rather than a transaction rollback event, is issued, because the flow has taken control of the error and terminated normally. In this case, if you need to distinguish errors, you can configure terminal events at appropriate nodes in the flow.

Event output options

Events are published to a topic, where they can be read by multiple subscribers. The topic name is of the form:

`$SYS/Broker/`*`brokerName`*`/Monitoring/`*`executionGroupName`*`/`*`flowName`*

The hierarchical structure allows subscribers to filter the events which they receive. One subscriber can receive events from all message flows in the broker, while another receives only the events from a single execution group.

Events do not participate in transactions. If transaction rollback occurs in a message flow, no events emitted by the message flow are rolled back, and all remain published.

Default monitoring configuration

If monitoring is activated for a message flow, and neither monitoring properties nor a monitoring profile configurable service have been configured for the flow, the default behavior is for transaction events to be emitted from each input node of the message flow. The events contain the bit stream of the input message.

Monitoring scenarios

Events can be used to support transaction monitoring, transaction auditing and business process monitoring.

Transaction monitoring and auditing

The events published by WebSphere Message Broker can be written to a transaction repository, creating an audit trail of the transactions that are processed by a broker. A transaction repository can be used for monitoring, auditing and replay of transactions. Bitstream data can be included so that failed transactions can be resubmitted. You can perform the following tasks to set up transaction monitoring and auditing.

Configure events for your transactions

In most cases bitstream information is not sufficient to allow querying of the logged transactions. Key fields and other correlation data can be extracted from the message payload and placed into the `wmb:applicationData/wmb:simpleContent` or `wmb:applicationData/wmb:complexContent` element of the event. The logging application or message flow can extract these fields and log them with the message bit stream.

Subscribe to the event topic and write events to a repository

You can create a message flow, or any WebSphere MQ application, that subscribes to the event topic and writes events to a relational database. The

details of the database schema depend on the requirements of your organization, for example the number of key fields and transaction IDs.

Business process monitoring

The events published by a broker can be monitored by WebSphere Business Monitor. Important fields in the message payload can be added to the events emitted by your message flows, allowing them to be monitored. You can use the following items to help you use WebSphere Business Monitor to monitor your message flows:

Message Driven bean
The events must be submitted to the CEI repository in order for WebSphere Business Monitor to monitor them. A message driven bean is supplied for this purpose. The message driven bean, which runs in WebSphere Application Server, subscribes to the event topic and writes events that match its subscription to the CEI repository as CBE events.

WebSphere Business Monitor Model
WebSphere Message Broker includes an example monitor model for use with WebSphere Business Monitor. This model demonstrates how to monitor transaction events and terminal events, including events that capture data from the input message and output message. Modify the model to match your actual events and message formats.

The following sample provides a Message Driven Bean and a WebSphere Business Monitor Model to help you use WebSphere Business Monitor to monitor events in your message flows:

- WebSphere Business Monitor

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

Deciding how to configure monitoring events for message flows

Decide whether to use monitoring properties, or a monitoring profile configurable service, to customize the events produced by a message flow.

If you want to customize events when you are designing the message flow, use monitoring properties. If you want to customize events after a message flow has been deployed, but without redeploying, use a monitoring profile. This topic gives information about both methods.

Using monitoring properties to configure events

Using the workbench Message Flow Editor, you can configure event sources on most nodes in a message flow. A node that supports the configuration of event sources has a Monitoring tab on its Properties view; use this tab to add events and to set their properties. When you deploy the message flow in a broker archive file, the monitoring properties are included as part of the message flow.

Key facts about monitoring properties:

- Use monitoring properties when you want to configure events at message flow design time.
- Monitoring properties apply only to the message flow in question. You can share monitoring properties between message flows only by creating reusable subflows.

- Monitoring properties are deployed in a BAR file as part of the message flow.
- You use the mqsichangeflowmonitoring command to activate the message flow to emit monitoring events.
- You can use the mqsichangeflowmonitoring command to enable or disable individual event sources in a message flow.
- To change any other properties of an event, you alter the monitoring properties in the flow, then redeploy the BAR file.

To configure monitoring events using monitoring properties, see this information:
"Configuring monitoring event sources using monitoring properties" on page 146.

Tip: Use the mqsireportflowmonitoring command to report a list of the names of the event sources for a message flow. You can then use these event source names in a mqsichangeflowmonitoring command to enable or disable individual event sources from the command line.

Using a monitoring profile to configure events

Using operational commands, you can create a monitoring profile configurable service directly on the broker, and associate it with one or more message flows.

Key facts about monitoring profiles:

- Use a monitoring profile when you want to configure events for a message flow that has already been deployed and for which no events have been configured.
- Use a monitoring profile to override the monitoring properties of a message flow that has already been deployed, as an alternative to redeploying the BAR file. The monitoring profile completely replaces all monitoring properties.
- A single monitoring profile can be applied to many message flows, as long as the message flows contain the same event sources.
- Monitoring profiles are created directly on the broker using the mqsicreateconfigurableservice command and the mqsichangeproperties command. They are not deployed in a BAR file.
- You use the mqsichangeflowmonitoring command to associate the monitoring profile with a message flow.
- You use the mqsichangeflowmonitoring command to activate the message flow to emit monitoring events.
- You can use the mqsichangeflowmonitoring command to enable or disable individual event sources in a message flow.

To configure monitoring events using monitoring properties, see this information:
"Configuring monitoring event sources using a monitoring profile" on page 149

Tip: Use the mqsireportflowmonitoring command to report a list of event sources for a message flow. You can then use the names of these event source in a subsequent mqsichangeflowmonitoring command to enable and disable individual event sources from the command line.

Tip: Use the mqsireportflowmonitoring command to report the monitoring profile for a message flow. You can edit the profile, to add or change event sources, then update it using the mqsichangeproperties command.

Tip: Use the mqsireportflowmonitoring command to create the equivalent monitoring profile for a message flow for which monitoring properties are

configured. You can edit the profile, then use it to create a new monitoring profile configurable service using the mqsicreateconfigurableservice and mqsichangeproperties commands. You can then associate the new profile with the original flow using the mqsichangeflowmonitoring command. You can use this technique to override the monitoring properties for a message flow, without having to edit the flow and redeploy the BAR file.

Configuring monitoring event sources using monitoring properties

In the Message flow Editor, use the Monitoring tab on the properties of a node to add one or more monitoring events.

Before you start:

Read the following topics:

- "Business-level monitoring" on page 141
- "Monitoring basics" on page 141

You must have a message flow that contains a node to which you want to add a monitoring event.

Creating events:

An event source is a point in a message flow from which a monitoring event can be emitted. Each event source has a set of properties that control the contents of the monitoring events that it emits.

1. Display the properties for the node.
2. Select the Monitoring tab.
3. Click **Add**.

   The Add entry window is displayed.
4. Complete the **Event Source** field.

   The field has a drop-down list of all events that can be defined for this node. The event source information is used to populate the attributes of the `wmb:eventPointData/wmb:messageFlowData/wmb:node` element of the event. When you have selected the event source, the corresponding value for Event Source Address is displayed as a read-only property.

   Tip: If you later decide to enable or disable events using the mqsichangeflowmonitoring command, you must specify a value for Event Source Address, not Event Name.
5. Complete the **Event Name** details; select either **Literal** or **Data location**.

   Every monitoring event has a name that is placed in the `wmb:eventPointData/wmb:eventIdentity/@wmb:eventName` attribute of the event. The default names are shown in the following table:

Event source	Default event name	Example
transaction.Start	*nodeLabel*.TransactionStart	MQInputTransactionStart
transaction.End	*nodeLabel*.TransactionEnd	MQInputTransactionEnd
transaction.Rollback	*nodeLabel*.TransactionRollback	MQInputTransactionRollback
terminal	*nodeLabel*.Terminal	JavaComputeOutTerminal

You can override the default in these ways:

- By specifying an alternative literal string.

- By specifying an XPath query; the query extracts the event name from a field in the input message. Click **Edit** to use the XPath Expression Builder.

You cannot use monitoring properties to configure transaction events on the "Collector node" on page 888. Use a monitoring profile instead; see "Configuring monitoring event sources using a monitoring profile" on page 149.

6. Optional: Complete the **Event Payload** section if the event is to contain selected data fields extracted from the message. Click **Add** to launch the Add Data Location dialog box. Take one of the following steps:
   - Type in the location (for example, `$LocalEnvironment/File/Name`); or
   - Click **Edit** to launch XPath Expression Builder.

   You can extract one or more fields from the message data and include it with the event. The fields can be simple or complex. Simple content is contained in the `wmb:applicationData/wmb:simpleContent` field of the event; complex data is contained in the `wmb:applicationData/wmb:complexContent` field.

   This facility is commonly used for communicating significant business data in a business event. If the event contains the input bit stream, this facility can also be used to extract key fields, allowing another application to provide an audit trail or to resubmit failed messages.
7. Optional: Select the **Include bitstream data in payload** field if the event is to capture message bitstream data.

   Content
   For a transaction event source, select from Headers, Body, All. For a terminal event source, the only available option is All.

   Encoding
   Select from base64, HexBinary and CData (the original text, without encoding).
8. Optional: Select the Correlation tab, to complete details for event correlation.
9. Complete the **Event Correlation** details; for information about correlation, see "Correlation and monitoring events" on page 1497.

Every monitoring event must contain at least one correlation attribute, and can contain up to three. If you do not specify any correlation information, the first event source in the message flow allocates a unique identifier that all later event sources in the same transaction will use.

a. Optional: Complete the **Local transaction correlator** details.

   Automatic
   The local correlator used by the most recent event for this invocation of the message flow will be used. If no local correlator exists yet, a new unique value will be generated.

   Specify location of correlator
   Enter a value, or click Edit to launch the XPath Expression Builder. The local correlator will be read from the specified location in the message tree. Ensure that the specified location contains a correlator value unique to this invocation of the message flow.

b. Optional: Complete the **Parent transaction correlator** details to extract a correlation field from the parent transaction.

   Automatic
   The parent correlator used by the most recent event for this invocation of the message flow will be used. If no parent correlator exists yet, no parent correlator will be used.

Specify location of correlator
Enter a value, or click Edit to launch the XPath Expression Builder. The parent correlator will be read from the specified location in the message tree. Ensure that the specified location contains a suitable value for the parent correlator.

c. Optional: Complete the **Global transaction correlator** details to extract a correlation field from a global transaction.

Automatic
The global correlator used by the most recent event for this invocation of the message flow will be used. If no global correlator exists yet, no global correlator will be used.

Specify location of correlator
Enter a value, or click Edit to launch the XPath Expression Builder. The global correlator will be read from the specified location in the message tree. Ensure that the specified location contains a suitable value for the global correlator.

10. Click **Finish**.

    The **Events** table in the Monitoring tab of the Properties view for the node is updated with details of the event that you just added; the event is enabled.
11. Optional: Disable the event.
12. Save the message flow.

Deploying monitoring properties:

1. When you have added all events to the flow, add the message flow to the broker archive (BAR) file and deploy the BAR file.

   Monitoring is inactive for the flow; deploying the BAR file does not activate it.
2. Activate monitoring for the flow by using the mqsichangeflowmonitoring -c command.

Updating monitoring properties:

The monitoring properties for a node show all monitoring events that are defined for that node. Edit the monitoring properties of a node to do these tasks:

- Enable or disable a monitoring event.
- Add, delete, or change the monitoring events for the node.

1. Right-click the node and select **Properties**.
2. Select the Monitoring tab.

   The monitoring events that you have previously defined are displayed.
3. Select or clear the **Enabled** check box for each event as appropriate.
   - To disable an event, clear the check box.
   - To enable an event, select the check box.
4. To add an event, click **Add**.
5. To delete an event, select the event, then click **Delete**.
6. To edit an event, select the event, then click **Edit**.

   The Add Event is displayed. For a description of options on this window, see "Creating events" on page 146.
7. Save the message flow.

Configuring monitoring event sources using a monitoring profile

You can create a monitoring profile and use the mqsichangeflowmonitoring command to configure your message flows to emit monitoring events.

Before you start:

Read the following topics:

- "Business-level monitoring" on page 141
- "Monitoring basics" on page 141

You must have a message flow that contains a node to which you want to add a monitoring event.

You can use XPath 1.0 expressions to configure a monitoring event.

Creating a monitoring profile:

First create a monitoring profile XML file. This is a file that lists the event sources in the message flow that will emit events, and defines the properties of each event.

Follow the guidance at "Monitoring profile" on page 1490 to create your monitoring profile XML file.

Applying a monitoring profile:

When you have created a monitoring profile XML file, follow these steps to apply it.

1. Use the mqsicreateconfigurableservice command to create a configurable service for the monitoring profile. In the following command example, replace *myBroker* with the name of your broker, and *myMonitoringProfile* with the name of your monitoring profile.

   ```
   mqsicreateconfigurableservice myBroker -c MonitoringProfiles
        -o myMonitoringProfile
   ```

2. Use the mqsichangeproperties command to associate your monitoring profile XML file with the configurable service. In the following command example, replace *myBroker* with the name of your broker, *myMonitoringProfile* with the name of your monitoring profile, and *myMonitoringProfile.xml* with the name of the monitoring profile XML file.

   ```
   mqsichangeproperties myBroker -c MonitoringProfiles -o myMonitoringProfile
        -n profileProperties -p myMonitoringProfile.xml
   ```

3. Use the mqsichangeflowmonitoring command to apply a monitoring profile configurable service to one or more message flows.

   - Apply a monitoring profile to a single message flow *messageflow1* in execution group *EG1*:

     ```
     mqsichangeflowmonitoring myBroker -e EG1
          -f messageflow1 -m myMonitoringProfile
     ```

   - Apply a monitoring profile to all message flows in all execution groups:

     ```
     mqsichangeflowmonitoring myBroker -g -j -m myMonitoringProfile
     ```

   Monitoring for the flow is inactive; applying the monitoring profile does not activate it.

4. Alternatively, use the broker archive editor to apply a monitoring profile configurable service to one or more message flows, by setting message flow property **Monitoring Profile Name**.

a. In the workbench, switch to the Broker Application Development perspective.
b. In the Broker Development view, right-click the BAR file, then click **Open with → Broker Archive Editor**.
c. Click the **Manage and Configure** tab.
d. Click the message flow on which you want to set the monitoring profile configurable service. The properties that you can configure for the message flow are displayed in the **Properties** view.
e. In the **Monitoring Profile Name** field, enter the name of a monitoring profile.
f. Save the BAR file.
g. Deploy the BAR file.

Monitoring for the flow is inactive; deploying the BAR file does not activate it.

5. Activate monitoring for the flow using the `mqsichangeflowmonitoring -c` command.

Updating a monitoring profile:

1. Follow the guidance at "Monitoring profile" on page 1490 to update your monitoring profile XML file.
2. Use the mqsichangeproperties command to update the configurable service to use the new XML file. For example:

```
mqsichangeproperties myBroker -c MonitoringProfiles -o myMonitoringProfile
     -n profileProperties -p myMonitoringProfile.xml
```

Activating monitoring

Use the mqsichangeflowmonitoring command to activate monitoring after you have configured monitoring event sources.

Before you start: You must have configured monitoring event sources by using either monitoring properties or a monitoring profile; see these topics:

"Configuring monitoring event sources using monitoring properties" on page 146

"Configuring monitoring event sources using a monitoring profile" on page 149

The use of a monitoring profile configurable service overrides the monitoring properties of a message flow.

If monitoring is activated for a message flow, and neither monitoring properties nor a monitoring profile configurable service have been configured for the flow, the default behavior is for transaction events to be emitted from each input node of the message flow. The events contain the bit stream of the input message.

Activating monitoring from the command line:

Use the `-c active` parameter. You can activate monitoring for all message flows in all execution groups, or specify the execution groups and message flows for which monitoring is to be activated.

- To activate monitoring for all message flows in all execution groups on Linux, UNIX, and Windows: Linux UNIX Windows

```
mqsichangeflowmonitoring myBroker -c active -g -j
```

To activate monitoring for all message flows in all execution groups on z/OS: z/OS

```
F MI10BRK,cm c=active,g=yes,j=yes
```

- To activate monitoring for all message flows in the default execution group on Linux, UNIX, and Windows: Linux UNIX Windows

```
mqsichangeflowmonitoring myBroker -c active -e default -j
```

To activate monitoring for all message flows in the default execution group on z/OS: z/OS

```
F MI10BRK,cm c=active,g=default,j=yes
```

- To activate monitoring for the myFlow message flow in the default execution group on Linux, UNIX, and Windows: Linux UNIX Windows

```
mqsichangeflowmonitoring myBroker -c active -e default -f myFlow
```

To activate monitoring for the myFlow message flow in the default execution group on z/OS: z/OS

```
F MI10BRK,cm c=active,g=default,f=myFlow
```

Deactivating monitoring from the command line:

Use the `-c inactive` parameter. You can deactivate monitoring for all message flows in all execution groups, or specify the execution groups and message flows for which monitoring is to be activated.

- To deactivate monitoring for all message flows in all execution groups on Linux, UNIX, and Windows: Linux UNIX Windows

```
mqsichangeflowmonitoring myBroker -c inactive -g -j
```

To deactivate monitoring for all message flows in all execution groups on z/OS: z/OS

```
F MI10BRK,cm c=inactive,g=yes,j=yes
```

- To deactivate monitoring for all message flows in the default execution group on Linux, UNIX, and Windows: Linux UNIX Windows

```
mqsichangeflowmonitoring myBroker -c inactive -e default -j
```

To deactivate monitoring for all message flows in the default execution group on z/OS: z/OS

```
F MI10BRK,cm c=inactive,g=default,j=yes
```

- To deactivate monitoring for the myFlow message flow in the default execution group on Linux, UNIX, and Windows: Linux UNIX Windows

```
mqsichangeflowmonitoring myBroker -c inactive -e default -f myFlow
```

To deactivate monitoring for the myFlow message flow in the default execution group on z/OS: z/OS

```
F MI10BRK,cm c=inactive,g=default,f=myFlow
```

Enabling and disabling event sources

When events have been configured for a message flow and deployed to the broker, you can enable and disable individual events. You can do this from the command line, without having to redeploy the message flow, or you can do this from the Message Flow Editor, in which case you must redeploy.

Enabling and disabling events from the command line:

Before you start: You must have previously configured events using either Message Flow Editor monitoring properties, or a monitoring profile configurable service.

Use the mqsichangeflowmonitoring to enable and disable event sources. For example:

```
mqsichangeflowmonitoring WBRK_BROKER
     -e default
     -f myMessageFlow
     -s "SOAP Input1.terminal.out,MQOutput1.terminal.in"
     -i enabled
mqsichangeflowmonitoring WBRK_BROKER
     -e default
     -f myMessageFlow
     -s "SOAP Input1.terminal.catch"
     -i disabled
```

You can enable or disable multiple events at once; the change of state takes place immediately.
If you configured events using monitoring properties, the change persists if the message flow is restarted, but is lost if the message flow is redeployed. To make the change permanent, you must also update the monitoring properties.

Tip: When specifying values for the -s parameter, use the **Event source address** property of the event, not the **Event name** property.

Tip: To find the list of configured event sources for a message flow, you can use the mqsireportflowmonitoring command. For example:

```
mqsireportflowmonitoring WBRK_BROKER
      –e default
      –f myMessageFlow
      –n
```

Tip: If you configured events using monitoring properties, you can see a list of the configured event sources in the Message Flow Editor:

1. Open the message flow using the Message Flow Editor.
2. Click the canvas.
3. Select the Monitoring tab in the Properties view.

   The monitoring events that you have previously defined are displayed.

Enabling and disabling events from the Message Flow Editor:

Before you start: You must have previously configured events using the Message Flow Editor monitoring properties.

Use the Message Flow Editor to enable and disable event sources.

1. Open the message flow using the editor.
2. Click the canvas.
3. Select the Monitoring tab in the Properties view. The monitoring events that you have previously defined are displayed.
4. Select or clear the **Enabled** check box for each event as appropriate.
   - To disable an event, clear the check box.
   - To enable an event, select the check box.

- To disable all events, click **Uncheck All**.

5. Save the message flow.
6. Rebuild and redeploy the BAR file.

Creating a monitoring model for use by WebSphere Business Monitor

Enable WebSphere Business Monitor to monitor WebSphere Message broker events.

Before you start: You must have configured monitoring for a message flow.

IBM WebSphere Business Monitor is comprehensive business activity monitoring software that provides a real-time view of your business processes and operations. It contains personalized business dashboards that calculate and display key performance indicators (KPIs) and metrics derived from business processes, business activity data, and business events from a wide range of information sources. This gives business users immediate actionable insight into their business operations to mitigate problems or take immediate advantage of opportunities, resulting in cost savings and increased revenues.

WebSphere Business Monitor collects data from a wide variety of sources, including WebSphere Message Broker message flows. The WebSphere Business Monitor sample in the Samples Gallery shows how to generate events from a message flow, create a corresponding monitor model, and use that model in WebSphere Business Monitor to view KPIs from the events in a dashboard; see WebSphere Business Monitor. You can view samples information only when you use the information center that is integrated with the Message Broker Toolkit or the online information center.

1. If an event contains data extracted from a message, and the data is complex, supply details of the structure of the data to WebSphere Business Monitor:
   a. Ensure that the complex data is modeled as a complex type in a message definition file within a message set in the WebSphere Message Broker toolkit.
   b. In the Application Development perspective, right-click the message set folder and select **Generate XML Schemas** to create a .zip file of XML Schemas from the definitions in the message set.
   c. Uncompress the file containing the XML schemas on the machine where you will be using the Monitoring Model Editor.
2. Export the `WMBEvent.xsd` file from an existing message set (or create a new message set, and then export it from there):
   a. In the Application Development perspective, right-click the message set and select **New → Message Definition File From → IBM Supplied Message**.
   b. In the window that is displayed, scroll down and click **Message Broker Monitoring Event**.
   c. Click **Finish**.
   d. Right-click the message set again, and select **Generate → XML Schemas**.
   e. In the window that is displayed, select **Export to an external directory** and then enter or browse to a directory.
   f. Click **Finish**. A .zip file containing file `WMBEvent.xsd` is created in the directory that you specified.

   The `WMBEvent.xsd` file gives Websphere Business Monitor details of the structure of the Message Broker event.

What to do next: This section outlines the steps that you need to take in WebSphere Business Monitor. See the documentation for that product for full details.

1. In the WebSphere Business Monitor development toolkit, create a Monitor Model.

   Import the WMBEvent.xsd schema and any schemas describing complex data that were exported from the WebSphere Message Broker Toolkit, and then create a monitor model. You will see errors, because the model is currently empty. You will also see a key, which you can rename, for example to LocalTransactionID.

2. Create WebSphere Business Monitor inbound events.

   A monitoring context definition is the term used by WebSphere Business Monitor to describe all the data that should be collected about an entity (such as a transaction or business process). Each runtime instance (referred to as a monitoring context instance) collects information from inbound events and stores this information in fields that represent the business measures that a monitoring context collects: metrics, counters, and stopwatches.

   You need to define an inbound event to describe each event source defined in your message flow that contains information that you want to monitor. For example, if your message flow has Transaction start, Transaction end and Transaction rollback event sources, you define an inbound event for each of these event sources.

   You normally want to define inbound events for these three event sources because they contain information that tells WebSphere Business Monitor when the start and end of the monitoring context instance is. You also define inbound events to describe any event sources downstream in the flow that contain data that you want to monitor, for example in the In terminal of the MQOutput node.Creating inbound events typically involves the following actions:

   a. Define event parts.

      Within the inbound event you define event parts. These are XML Schema definition (XSD) types that provide information about the structure of part of an event. So for a WebSphere Message Broker event you need to define some event parts to describe the different parts of the event that you want to monitor data from. For a description of the event structure, see "The monitoring event" on page 1495.

      As a minimum you need to define an event part for the event described by type wmb:event. To monitor data about the source of the event (information about the message flow name, broker name) you also need to define an event part for the eventPointData section described by type wmb:eventPointData. You might also then want to define event parts to describe message payload data from the applicationData section of the event.

   b. Define a correlation expression.

      You typically correlate events on fields from the eventCorrelation section in the WebSphere Message Broker event. So, for example, you could correlate events using the localTransactionId field from the WebSphere Message Broker event.

      You also need to define whether the event should create a new monitoring context; create this for a Transaction start event.

   c. Define a filter condition.

      Optionally you can set a filter condition. For example you might want to filter events for a specific broker, execution group, and message flow.

d. Define the event sequence path.

   Optionally you select a field in the inbound event which can be used to set the order in which the inbound events are processed. For example you could use creationTime from the WebSphere Message Broker event.

e. Complete the key.

   The key uniquely identifies a monitoring context instance. You can select any value for the key; for a Websphere Message Broker event a typical value is the localTransactionId field from the Websphere Message Broker event.

3. Define the metrics.

   Having defined inbound events you can then define your metrics. Metrics hold data from the event in a monitoring context.

   You might want to define metrics that hold event source data from the eventPointData section of the WebSphere Message Broker event, for example broker name, or message flow name. You can also define metrics that hold event sequencing information, for example TimeStarted and TimeEnded metrics which hold the creationTime for Transaction start and Transaction end or Transaction rollback events.

   In addition metrics can be defined to hold application data from the WebSphere Message Broker event.

Reporting monitoring settings

Use the mqsireportflowmonitoring command to report monitoring settings for a flow.

The following examples show how to report monitoring options for a broker called `BROKER1`, execution group `default`, and message flow `PurchaseOrder`.

Report configured events for a message flow:

Issue the following command on Linux, Unix, or Windows:

Linux UNIX Windows

```
mqsireportflowmonitoring BROKER1 –e default –f PurchaseOrder –n
```

Issue the following command on z/OS: z/OS

```
F MI10BRK,rm BROKER1,e='default',f='PurchaseOrder,n='yes''
```

Report all possible events for a message flow:

Issue the following command on Linux, Unix, or Windows:

Linux UNIX Windows

```
mqsireportflowmonitoring BROKER1 –e default –f PurchaseOrder –a
```

Issue the following command on z/OS: z/OS

```
F MI10BRK,rm e='default',f='PurchaseOrder,a='yes'
```

Report specified events for a message flow:

Issue the following command on Linux, Unix, or Windows:

Linux UNIX Windows

```
mqsireportflowmonitoring BROKER1 –e default –f PurchaseOrder
     -s "MQInput1.transaction.Start,MQOutput1.terminal.catch"
```

Issue the following command on z/OS: z/OS

```
F MI10BRK,rm e='default',f='PurchaseOrder',
     s="MQInput1.transaction.Start,MQOutput1.terminal.catch"
```

Tip: When specifying values for the -s parameter, use the **Event source address** property of the event, not the **Event name** property.

Export a message flow's monitoring profile:

Use the following command to export a profile to file `myMonProf.xml`

Issue the following command on Linux, Unix, or Windows:
Linux UNIX Windows

```
mqsireportflowmonitoring BROKER1 -e default -f PurchaseOrder -x -p myMonProf.xml
```

Issue the following command on z/OS: z/OS

```
F MI10BRK,rm e='default',f='PurchaseOrder',x='yes',p='myMonProf.xml'
```

If monitoring for a message flow is configured using monitoring properties, rather than a monitoring profile configurable service, the command creates and returns the equivalent monitoring profile XML file.

Tip: This is an alternative to using an XML editor to create a monitoring profile XML file.

Message flow accounting and statistics data

Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution.

These reports are not the same as the publish/subscribe statistics reports that you can generate. The publish/subscribe statistics provide information about the performance of brokers and the throughput between the broker and clients that are connected to the broker. Message flow accounting and statistics reports provide information about the performance and operating details of a message flow execution.

Message flow accounting and statistics data records dynamic information about the runtime behavior of a message flow. For example, it indicates how many messages are processed and how large those messages are, as well as processor usage and elapsed processing times. The broker collects the data and records it in a specified location when one of a number of events occurs (for example, when a snapshot interval expires or when the execution group you are recording information about stops).

The broker takes information about statistics and accounting from the operating system. On some operating systems, such as Windows, Linux, and UNIX, rounding can occur because the system calls that are used to determine the processor times are not sufficiently granular. This rounding might affect the accuracy of the data.

The following restrictions apply to data collection:

- If the message flow starts with a SCADAInput node or a Real-timeInput node, no data is collected (and no error is reported).

- Data relating to the size of messages is not collected for WebSphere Adapters nodes (for example, the SAPInput node), the FileInput node, the JMSInput node, or any user-defined input node that does not create a message tree from a bit stream.

Collecting message flow accounting and statistics data is optional; by default it is switched off. To use this facility, request it on a message flow or execution group basis. The settings for accounting and statistics data collection are reset to the defaults when an execution group is redeployed. Previous settings for message flows in an execution group are not passed on to the new message flows deployed to that execution group. Data collection is started and stopped dynamically when you issue the mqsichangeflowstats command; you do not need to change the broker or the message flow, or redeploy the message flow, to request statistics collection.

You can activate data collection on both your production and test systems. If you collect the default level of statistics (message flow), the effect on broker performance is minimal. However, collecting more data than the default message flow statistics can generate high volumes of report data that might affect performance slightly.

When you plan data collection, consider the following points:
- Collection options
- Accounting origin
- Output formats

You can find more information about how to use accounting and statistics data to improve the performance of a message flow in this developerWorks® article on message flow performance.

Message flow accounting and statistics collection options

The options that you specify for message flow accounting and statistics collection determine what information is collected. You can request two types of data collection: Snapshot data and archive data.

- Snapshot data is collected for an interval of approximately 20 seconds. The exact length of the interval depends on system loading and the level of current broker activity. You cannot modify the length of time for which snapshot data is collected. At the end of this interval, the recorded statistics are written to the output destination and the interval is restarted.
- Archive data is collected for an interval that you have set for the broker on the mqsicreatebroker or mqsichangebroker command. You can specify an interval of between 10 and 14400 minutes, the default value is 60 minutes. At the end of this interval, the recorded statistics are written to the output destination and the interval is restarted.

  An interval is prematurely expired and restarted when any of the following events occur:
  - The message flow is redeployed.
  - The set of statistics data to be collected is modified.
  - The broker is shut down.

  This preserves the integrity of the data already collected when that event occurs.

  z/OS On z/OS, you can set the command parameter to 0, which means that the interval is controlled by an external timer mechanism. This support is provided by the Event Notification Facility (ENF), which you can use instead of the broker command parameter if you want to coordinate the expiration of this timer with other system events.

You can request snapshot data collection, archive data collection, or both. You can activate snapshot data collection while archive data collection is active. The data recorded in both reports is the same, but is collected for different intervals. If you activate both snapshot and archive data collection, be careful not to combine information from the two different reports, because you might count information twice.

You can use the statistics generated for the following purposes:

- You can record the load that applications, trading partners, or other users put on the broker. This allows you to record the relative use that different users make of the broker, and perhaps to charge them accordingly. For example, you could levy a nominal charge on every message that is processed by a broker, or by a specific message flow.

  Archive data provides the information that you need for a use assessment of this kind.
- You can assess the execution of a message flow to determine why it, or a node within it, is not performing as you expect.

  Snapshot data is appropriate for performance assessment.
- You can determine the route that messages are taking through a message flow. For example, you might find that an error path is taken more frequently than you expect and you can use the statistics to understand when the messages are routed to this error path.

  Check the information provided by snapshot data for routing information; if this is insufficient for your needs, use archive data.

Message flow accounting and statistics accounting origin

Accounting and statistics data can be accumulated and reported with reference to an identifier associated with a message in a message flow. This identifier is the accounting origin, which provides a method of producing individual accounting and statistics data for multiple accounting origins that generate input to message flows. The accounting origin can be a fixed value, or it can be dynamically set according to your criteria.

For example, if your broker hosts a set of message flows associated with a particular client in a single execution group, you can set a specific value for the accounting origin for all these flows. You can then analyze the output provided to assess the use that the client or department makes of the broker, and charge them accordingly.

If you want to track the behavior of a particular message flow, you can set a unique accounting origin for this message flow, and analyze its activity over a specified period.

To make use of the accounting origin, you must perform the following tasks:

- Activate data collection, specifying the correct parameter to request basic support (the default is none, or no support). For details, see mqsichangeflowstats command.
- Configure each message flow for which you want a specific origin to include ESQL statements that set the unique value that is to be associated with the data collected. Data for message flows for which a specific value has not been set are identified with the value `Anonymous`.

  The ESQL statements can be coded in a Compute, Database, or Filter node.

  You can configure the message flow either to set a fixed value, or to determine a dynamic value, and can therefore create a very flexible method of recording sets

of data that are specific to particular messages or circumstances. For more information, refer to “Setting message flow accounting and statistics accounting origin” on page 659.

You can complete these tasks in either order; if you configure the message flow before starting data collection, the broker ignores the setting. If you start data collection, specifying accounting origin support, before configuring the message flow, all data is collected with the Accounting Origin set to `Anonymous`. The broker acknowledges the origin when you redeploy the message flow. You can also modify data collection that has already started to request accounting origin support from the time that you issue the command. In both cases, data that has already been collected is written out and collection is restarted.

When data has been collected, you can review information for one or more specific origins. For example, if you select XML publication messages as your output format, you can start an application that subscribes to the origin in which you are interested.

Output formats for message flow accounting and statistics data

When you collect message flow statistics, you can choose the output destination for the data.

Select one of the following destinations:
- User trace
- XML publication
- SMF

Statistics data is written to the specified output location in the following circumstances:

- When the archive data interval expires.
- When the snapshot interval expires.
- When the broker shuts down. Any data that has been collected by the broker, but has not yet been written to the specified output destination, is written during shutdown. It might therefore represent data for an incomplete interval.
- When any part of the broker configuration is redeployed. Redeployed configuration data might contain an updated configuration that is not consistent with the existing record structure (for example, a message flow might include an additional node, or an execution group might include a new message flow). Therefore the current data, which might represent an incomplete interval, is written to the output destination. Data collection continues for the redeployed configuration until you change data collection parameters or stop data collection.
- When data collection parameters are modified. If you update the parameters that you have set for data collection, all data that is collected for the message flow (or message flows) is written to the output destination to retain data integrity. Statistics collection is restarted according to the new parameters.
- When an error occurs that terminates data collection. You must restart data collection yourself in this case.

User trace

You can specify that the data that is collected is written to the user trace log. The data is written even when trace is switched off. The default output destination for accounting and statistics data is the user trace log. The data is written to one of the following locations:

Windows **Windows**

If you set the work path by using the **-w** parameter of the mqsicreatebroker command, the location is `workpath\log`.

If you have not specified the broker work path, the location is:

- On Windows:`%ALLUSERSPROFILE%\Application Data\IBM\MQSI\common\log`, where `%ALLUSERSPROFILE%` is the environment variable that defines the system working directory. The default directory depends on the operating system:
  - On Windows XP and Windows Server 2003: `C:\Documents and Settings\All Users\Application Data\IBM\MQSI\common\log`
  - On Windows Vista and Windows Server 2008: `C:\ProgramData\IBM\MQSI\common\log`

  The value might be different on your computer.

Linux UNIX **Linux and UNIX**

`/var/mqsi/common/log`

z/OS **z/OS**

`/component_filesystem/log`

XML publication

You can specify that the data that is collected is published. The publication message is created in XML format and is available to subscribers registered in the broker network that subscribe to the correct topic.

The topic on which the data is published has the following structure:

`$SYS/Broker/`*`brokerName`*`/StatisticsAccounting/`*`recordType`*`/`*`executionGroupLabel`*`/`*`messageFlowLabel`*

The variables correspond to the following values:

brokerName
: The name of the broker for which statistics are collected.

recordType
: Set to `SnapShot` or `Archive`, depending on the type of data to which you are subscribing. Alternatively, use # to register for both snapshot and archive data if it is being produced. This value is case sensitive and must be entered as `SnapShot`.

executionGroupLabel
: The name of the execution group for which statistics are collected.

messageFlowLabel
: The label on the message flow for which statistics are collected.

Subscribers can include filter expressions to limit the publications that they receive. For example, they can choose to see only snapshot data, or to see data that is collected for a single broker. Subscribers can specify wild cards (+ and #) to receive publications that refer to multiple resources.

The following examples show the topic with which a subscriber registers to receive different sorts of data:

- Register the following topic for the subscriber to receive data for all message flows running on *BrokerA*:

  $SYS/Broker/*BrokerA*/StatisticsAccounting/#

- Register the following topic for the subscriber to receive only archive statistics that relate to a message flow *Flow1* running on execution group *Execution* on broker *BrokerA*:

  $SYS/Broker/*BrokerA*/StatisticsAccounting/Archive/*Execution*/*Flow1*
- Register the following topic for the subscriber to receive both snapshot and archive data for message flow *Flow1* running on execution group *Execution* on broker *BrokerA*

  $SYS/Broker/*BrokerA*/StatisticsAccouting/#/*Execution*/*Flow1*

For help with registering your subscriber, see Message display, test and performance utilities SupportPac (IH03).

SMF

On z/OS, you can specify that the data collected is written to SMF. Accounting and statistics data uses SMF type 117 records. SMF supports the collection of data from multiple subsystems, and you might therefore be able to synchronize the information that is recorded from different sources.

To interpret the information that is recorded, use any utility program that processes SMF records.

Message flow aggregation

Aggregation is the generation and fan-out of related requests that are derived from a single input message, and the fan-in of the corresponding replies to produce a single aggregated reply message.

The initial request that is received by the message flow, representing a collection of related request items, is split into the appropriate number of individual requests to satisfy the subtasks of the initial request. This process is known as *fan-out*, and it is provided by a message flow that includes aggregation nodes.

Replies from the subtasks are combined and merged into a single reply, which is returned to the original requester (or another target application) to indicate completion of the processing. This process is known as *fan-in*, and it is also provided by a message flow that includes aggregation nodes.

A message aggregation is initiated by a message flow that contains the AggregateControl node followed by an AggregateRequest node. The responses are aggregated back together using a flow that contains the AggregateReply node. The aggregation nodes work correctly only for transports that use a request/reply model; for example, WebSphere MQ Enterprise Transport.

WebSphere Message Broker provides three message flow nodes that support aggregation:

- The AggregateControl node
- The AggregateRequest node
- The AggregateReply node

When you include these nodes in your message flows, the multiple fan-out requests are issued in parallel from within a message flow. The standard operation of the message flow is for each node to perform its processing in sequence.

If you use WebSphere MQ Enterprise Transport, the responses that are received by the fan-in flow must be valid reply messages that contain the reply identifier. You must set the reply identifier to the value of the message in the request message's message descriptor (MQMD), then store the reply identifier in the correlation identifier field (CorrelId) of the MQMD. If the CorrelId is set to MQCI_NONE, the AggregateReply node issues an error because the reply message is not valid, and cannot be matched to a request message.

You can also use these aggregation nodes to issue requests to applications outside the broker environment. Messages can be sent asynchronously to external applications or services; the responses are retrieved from those applications, and the responses are combined to provide a single response to the original request message.

These nodes can help to improve response time because slow requests can be performed in parallel, and they do not need to follow each other sequentially. If the subtasks can be processed independently, and they do not need to be handled as part of a single unit of work, they can be processed by separate message flows.

You can design and configure a message flow that provides a similar function without using the aggregation nodes, by issuing the subtask requests to another application (for example, using the HTTPRequest node), and recording the results of each request in the local environment. After each subtask has completed, merge the results from the LocalEnvironment in a Compute node, and create the combined response message for propagating to the target application. However, all the subtasks are performed sequentially, and they do not provide the performance benefits of parallel operation that you can achieve if you use the aggregation nodes.

Examples of aggregation flows that use the aggregation nodes are provided in the following samples:

- Aggregation
- Airline Reservations

The Aggregation sample demonstrates a simple four-way aggregation, and the Airline Reservations sample simulates requests that are related to an airline reservation service, and illustrates the techniques that are associated with aggregation flows.

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

The aggregation nodes store state for aggregations on WebSphere MQ queues. By default, the following storage queues are used:

- SYSTEM.BROKER.AGGR.CONTROL
- SYSTEM.BROKER.AGGR.REPLY
- SYSTEM.BROKER.AGGR.REQUEST
- SYSTEM.BROKER.AGGR.UNKNOWN
- SYSTEM.BROKER.AGGR.TIMEOUT

However, you can create alternative storage queues and use an aggregation configurable service to specify which queues are to be used by the node. For more information, see “Configuring the storage of events for aggregation nodes” on page 653.

By default, the timeout on the AggregateControl node is set to 0. If you do not specify a timeout (or if you leave it set to 0), aggregation requests that WebSphere MQ stores are never deleted unless all reply messages are returned. This situation might lead to a gradual build up of messages on the internal queues. Set the timeout to a value greater than zero to ensure that requests are removed and that queues do not fill up with redundant requests. It is good practice to set the timeout value to a large value, for example, 86400 seconds (24 hours), so that the queues clear old aggregation messages even if timeouts are not required or expected.

You can set the timeout either by setting the Timeout property on the AggregateControl node, or by using an Aggregation configurable service and specifying the timeoutSeconds property. For more information, see "Setting timeouts for aggregation" on page 676.

The aggregation nodes use WebSphere MQ message expiry to manage timeout of messages. For message expiry to work, the aggregation nodes must browse the message queues. The aggregation nodes browse the queues automatically to ensure that expired messages are processed.

z/OS On z/OS, you can configure WebSphere MQ to run a scavenger process that browses the queues instead of the aggregation nodes. To enable the scavenger, set the broker's queue manager property EXPRYINT to a value of 5 seconds. If you do not set EXPRYINT, or set it to a value higher than 10 seconds, the aggregation nodes revert to browsing the aggregation queues automatically.

Message collections

A message collection is a single message that contains multiple messages derived from one or more sources.

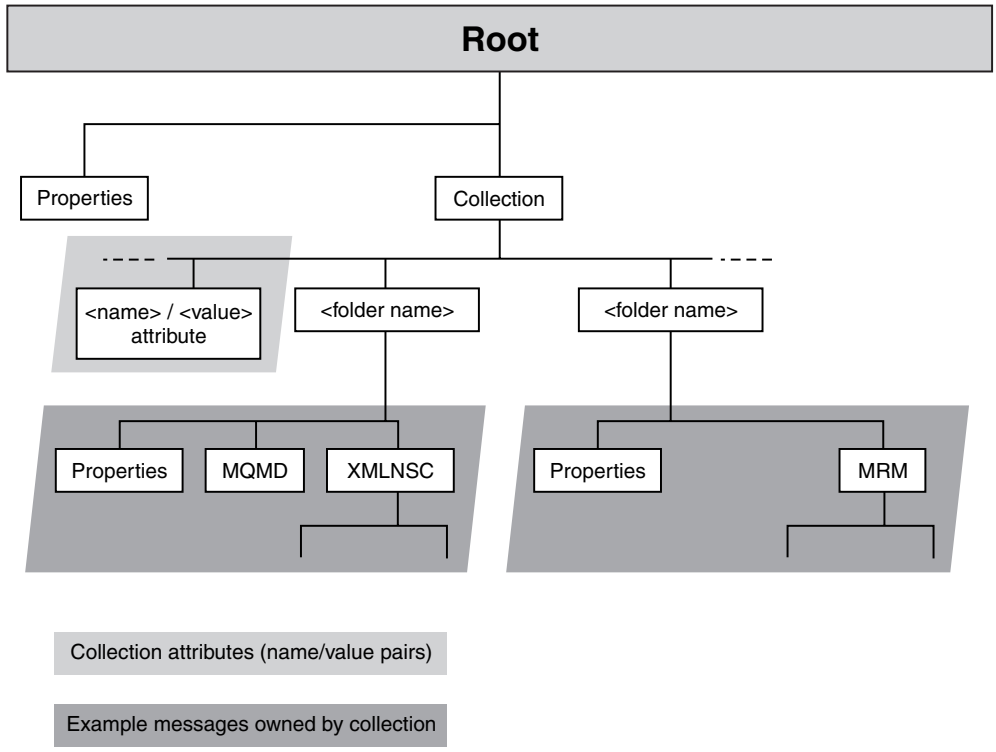
You can use a Collector node to group together messages from one or more sources into a message collection, so that they can be processed together by downstream nodes. You can also manually build a message collection using a JavaCompute node.

Structure of a message collection

A message collection tree contains sub-trees that hold the content of the individual messages received by the Collector node. The message assembly that is propagated from the Collector node to other nodes in your message flow contains the following four components:

- Message (including transport headers)
- Local environment
- Global environment
- Exception list

The following figure shows an example of the message tree structure of a message collection.



The message collection in this example contains two messages, one received from WebSphere MQ, and one from a file input source.

A message collection has a Properties header and a single folder element named Collection. A message collection can also have zero or more attributes that are name/value pairs; the name of an attribute must be unique within a message collection. These are shown as `<name> / <value>` in the figure. A standard attribute for the message collection is an attribute called *`CollectionName`*. If you use a Collector node to generate a message collection, the value for the collection name is generated based on the values you configure in the node. The collection name attribute is not compulsory.

Within the Collection folder in the message collection tree structure are folders, shown as `<folder name>` in the diagram. These folders contain the message tree of each message added to the message collection. Each of these folders has a name, but this name does not have to be unique within the message collection. The value for the `<folder name>` is derived from the source of the input message.

Nested message collections are not permitted. You cannot therefore use a message collection as a source message for another message collection. For example, If you attempt to pass a message collection to a input terminal on a Collector node, an error is generated.

The LocalEnvironment, Environment and ExceptionList trees are not included in the structure, but are instead carried separately as a part of the message assembly. There is no concept of a LocalEnvironment associated with each message in a message collection.

Generating a message collection using a Collector node

You can use the Collector node to make multiple synchronous or asynchronous requests in parallel. The results of these requests can be joined together downstream if required. This is different from the behavior of the aggregation nodes where there is a fixed pattern of request/response and where reply messages are grouped by request id. In contrast, the collector node does not need an initial fan-out stage and can group together unrelated input messages by correlating their content. You can configure dynamic input terminals on a Collector node to receive messages from different sources. You can also configure properties on the Collector node, known as event handlers, to determine how messages are added to a message collection, and when a message collection is complete.

Processing a message collection

A message collection is supported by the following nodes only:

- Compute
- JavaCompute

Errors are generated by other nodes if they receive a message collection.

You can use ESQL or XPath expressions to access the content of messages in a message collection by referencing the folder names preceded by `InputRoot.Collection`. To access the contents of a message in a message collection using ESQL you can use code similar to the following ESQL:

```
InputRoot.Collection.folder1.XMLNSC
```

In XPath, the root element is the body of the message. The root element for a message collection is the Collection element. Therefore, to access the contents of a message in a message collection using XPath, you must use an expression similar to the following XPath:

```
/folder1/XMLNSC
```

Examples of XPath expressions that you can use to access the message collection are:

- `/*`: returns a list of all the messages in the message collection.
- `/@*`: returns a list of all the attributes of the message collection.
- `/@Name`: returns the value of the attribute `Name`.

You might not be able to determine the order of the messages within a message collection. If you generate a message collection using the Collector node, the messages are arranged in the same order as the messages arrived at the node.

Converting data with message flows

Convert data that your message flows are transferring between different environments by using WebSphere MQ or WebSphere Message Broker facilities.

Data conversion is the process by which data is transformed from the format recognized by one operating system into the format recognized by a second operating system with different characteristics such as numeric order.

If you are using a network of systems that use different methods for storing numeric values, or you need to communicate between users who view data in different code pages, you must consider how to implement data conversion.

Code page conversions

Code page conversion might be required for one or more of the following reasons:

- ASCII versus EBCDIC
- Code pages that are specific to national language
- Code pages that are specific to operating systems

In WebSphere MQ, these factors are handled by the CCSID field in the MQMD header. For more information about the MQMD header, see "MQMD - Message descriptor" in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online. For more information about code page support, see "Code page conversion", also in the *Application Programming Reference* section.

Encoding

Encoding (byte order) conversion might be required for one or both of the following reasons:

- Big endian versus little endian

  Endian is an attribute of data that describes whether it is stored in computer memory or transmitted with the most significant byte first (big endian) or last (little endian).

- Floating point number representations

In WebSphere MQ, these factors are handled by the Encoding field in the MQMD header. For more information about the MQMD header, see "MQMD - Message descriptor" in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online. For more information about encoding, see "Machine encoding", also in the *Application Programming Reference* section.

If you are configuring a message flow to receive messages:

- Messages received across a WebSphere MQ protocol that uses WebSphere MQ headers, contain code page encoding characteristics in the MQMD header, and optionally in other WebSphere MQ headers.
- Messages received across protocols that do not use WebSphere MQ headers do not include these characteristics. Configure these characteristics by using properties on the nodes in your message flows. For example, set the Message coded character set ID and Message encoding properties on the FileInput node.

If you are configuring a message flow to send messages to other applications or systems:

- Messages sent across a WebSphere MQ protocol contain code page encoding characteristics in the MQMD header, and optionally in other WebSphere MQ headers.
- Messages sent across protocols that do not use WebSphere MQ headers must be modified to include these characteristics in the Properties folder in the logical message tree structure. The parser called by the output node uses these values to generate the correct bit stream.

When you use WebSphere Message Broker, you can use the data conversion facilities of WebSphere Message Broker, WebSphere MQ, or both.

WebSphere Message Broker facilities

You can model your messages in the MRM through the workbench.

Predefined elements of the messages are converted according to their type and physical layer characteristics. For further details, see Configuring physical properties. You can also use self-defining messages. You can then use the Compute, JavaCompute, or PHPCompute node to configure encoding and CCSIDs. You do not need WebSphere MQ data conversion exits.

- String data is converted according to the CCSID setting.
- Decimal integer and float extended decimal types are converted according to the CCSID setting.
- Decimal integer and float (other physical data types) are converted according to the Encoding setting.
- Binary and Boolean data is not converted.

WebSphere Message Broker can also convert the WebSphere MQ headers for which parsers are provided.

When you use WebSphere Message Broker facilities, the whole message is not converted to the specified encoding and CCSID: you can specify a different encoding, or CCSID, or both, in each header to perform a different conversion for the following part of the message. The encoding and CCSID in the last header defines the values for the message body.

WebSphere MQ facilities

Headers and message body are converted according to the values set in the appropriate MQMD fields, and other header format names. You might need to set up data conversion exits to convert the body of your messages.

When you use WebSphere MQ facilities, the whole message is converted to the specified encoding and CCSID, according to the setting of the format in the WebSphere MQ header.

For more detail about data conversion using WebSphere MQ facilities, see "Data conversion" in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.

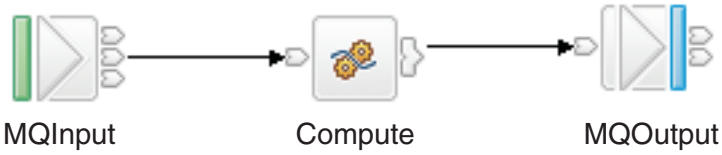
User exits

A user exit is user-provided custom software, written in C, to track data passing through message flows.

User-provided functions can be invoked at specific points during the life cycle of a message while it passes through the message flow, and can invoke utility functions to query information about the point in the flow, and the contents of the message assembly. The utility function can also modify certain parts of the message assembly. For more information about using user exits, see Why use a user exit?.

The user exits can be invoked when one or more of the following events occur:

- The end of a unit-of-work (UOW) or transaction (COMMIT or ROLLBACK).
- A message passes between two nodes.
- A message is successfully enqueued or sent to a transport in an output, reply, or request node.
- A message is dequeued or received in an input, response, or TimeoutNotification node.



In the basic message flow shown here, you can track messages at three levels:
- Transaction level
- Node level
- Input or output level

At the transaction level, you can track the following events:
- Messages being read into the flow
- Completion of the transaction

At the node level, you can track the following events:
- A message passing from one node to another
- Completion of processing for one node

At the message input or output level, you can track the following events:
- Messages being read into the flow
- Messages being written from the flow

Therefore, you can track five different types of event, which occur in the following sequence:

1. A message is dequeued from the input source (read into the flow).
2. A message is propagated to the node for processing.
3. A request message is sent to the output node's transport, and transport-specific destination information is written to "WrittenDestination" in the LocalEnvironment.
4. Node processing is completed.
5. The transaction ends.

Getting started with Quick Start wizards

A Quick Start wizard sets up the basic resources that are required to develop a broker application. The wizard sets up and gives names to containers for the resources in which you then develop your application.

The topics in this section describe how to use the Quick Start wizards.

Concept topics:

- “Quick Start wizards overview” on page 169

Task topics:

- “Creating an application from scratch” on page 169
- “Creating an application based on WSDL or XSD files” on page 170
- “Creating an application based on an existing message set” on page 172
- “Creating an application that uses WebSphere Adapters” on page 173

- "Creating an application by using the Configure New Web Service Usage wizard" on page 173

Quick Start wizards overview

You can use a Quick Start wizard to set up the basic resources that are required to develop a broker application. The wizard sets up and gives names to containers for the resources in which you then develop an application.

The resources that you can set up are described in the following list.

Message flow project
A specialized container in which you create and maintain all the resources that are associated with one or more message flows.

Message set project
A specialized container in which you create and maintain all the resources that are associated with a message set.

Message set
A container for grouping messages and associated message resources (elements, types, groups).

Message flow
A container for a sequence of processing steps that run in the broker when an input message is received.

Working set
A specialized container in which you can group related application projects so that you limit the number of resources that are displayed in the Broker Development view.

The following Quick Start wizards and links are available.

- **Start from scratch**, described in "Creating an application from scratch"
- **Start from WSDL and/or XSD files**, described in "Creating an application based on WSDL or XSD files" on page 170
- **Start from existing message set**, described in "Creating an application based on an existing message set" on page 172
- **Start from adapter connection**, described in "Creating an application that uses WebSphere Adapters" on page 173

Creating an application from scratch

Use the Start from scratch wizard to create the basic resources that are required to develop a broker application.

The Start from scratch wizard creates a message flow project, a message set project, sets up the project dependency, creates a message set and, optionally, creates a message flow and working set. To create these resources, complete the following steps:

1. Switch to the Broker Application Development perspective.
2. Open the Start from scratch wizard by using one of the following methods:
   - In the Projects section of the Broker Development view, if no projects are listed, the Quick Starts links are displayed.

     Click **Start from scratch**.
   - In the Projects section toolbar, click **Quick Starts...**, the Quick Starts links are displayed.

Click **Start from scratch**.

The **New Message Broker Application** panel of the wizard is displayed. In this panel, you can type the names of the basic resources that are required to develop a broker application.

3. Type into the appropriate fields, the names of the message flow project, the message set project, the message set, the message flow, and the name of the working set that contains the two new projects. Default names of the message flow project, the message flow, and the working set are already displayed in the appropriate fields, but you can edit these fields by typing your own names for these resources.

   You can change any of the names that are displayed by typing into the appropriate field the name that you want. You can also clear either of the check boxes that relate to the creation of a new message flow or a new working set; if you clear either of the check boxes, you cannot enter text into the associated name field, and the associated resource file is not created.
4. Click **Next**. The **Message set Physical Formats** panel is displayed. The panel lists three physical formats: **XML documents**, **Binary data** (Custom Wire Format), and **Text data** (TDS Format).
5. Select one or more of the check boxes to describe the type of message data that you want to process. If you do not select a check box, **XML documents** is selected by default.
6. Click **Finish** to complete the task. The Start from scratch wizard closes.

The wizard creates a message flow project, message set project, message set, and, optionally, a message flow, with the names that you have specified. It also creates, optionally, a new working set, with the name that you have specified. The working set contains all the resources you have created, and the Broker Development view shows the new working set as the active working set. If you have chosen not to create a new working set, the projects are created in the active working set currently shown in the Broker Development view.

The XML, CWF, or TDS formats are created with default names for the message set.

The message flow, if created, is opened in the message flow editor.

Next: If you have created a message flow, you can now go on to "Defining message flow content" on page 268.

Creating an application based on WSDL or XSD files

Create a new application that is based on existing WSDL or XSD files.

1. Switch to the Broker Application Development perspective.
2. Open the New Message Broker Application wizard.
   a. Click **File** → **New** → **Project**, or right-click anywhere in the Broker Development view, and click **New> Project**. The New Project window opens.
   b. Double-click **Message Brokers**. A list of wizards is displayed.
   c. Select **Start from WSDL and/or XSD files**, and click **Next**.

   The first panel of the New Message Broker Application wizard is displayed.
3. Set up the basic resources that are required to develop a broker application that uses existing WSDL and XSD files as a starting point.

- Type the name of your new application in the **Message flow project name** field.

  The name that you type is also displayed in the **Message set project name** and **Message set name** fields, but with 'MessageSet' appended.

  Similarly, the name that you type is also displayed in the **Message flow name** field (with 'Flow' appended), and in the **Working set name** field.
- Click **Next**.

  You can change any of the names that are displayed by typing into the appropriate field the name that you want. You can also clear either of the check boxes that relate to the creation of a new message flow or a new working set; if you do this, you cannot enter text into the associated name field.

4. Select the WSDL or XSD files that you want to use as the initial contents of the message set.
   - To choose WSDL or XSD files that exist in your workspace, click **Use resources from the workspace**.

     You are presented with a list of resources from which you can choose. Resources are filtered to show resources only in the active working set.
   - To choose WSDL or XSD files that exist outside your workspace, click **Use external resources** and type a directory name in the **From directory** field. Click **Browse**.

     You are presented with a list of the items in that directory. Make your choice from this list.

   In both cases, a two-pane view is displayed. On the left side, containers (for example, projects, folders, and message sets) are displayed. On the right side, the contents of these containers are shown. Depending on which button you clicked, either a workspace view or a file system view of the resources is displayed.

   If the only use of the XSD file is from the WSDL bindings, you do not have to select an XSD file that a selected WSDL files depends on.

   The view incorporates an option that allows you to copy the source file into the `importFiles` directory of the message set.

   You can use this option as follows:
   - If you choose only WSDL files, you can select the check box.
   - If you choose only XSD files, the option is automatically selected and the check box is greyed out. If you subsequently select a WSDL file, the check box is enabled but the selection state is not changed; that is, the check box remains selected.
   - Regardless of what you select, if the `importFiles` folder exists in the message set project after the import, it is collapsed.
   - If you import only WSDL files, the wizard sets the default message domain to SOAP.
5. Click **Next**. If you selected one or more WSDL files, the WSDL files that you selected are shown in a check box tree, with the acceptable bindings for each file shown as children.
6. (Optional) Select one or more bindings for each of the WSDL files that you selected. If you do not select at least one binding for each WSDL file, an error message is displayed and the **Next** and **Finish** buttons are disabled.
7. Click **Next**. If you selected one or more XSD files, the XSD files that you selected are displayed in the next pane, with the global elements for each file shown as children.

8. (Optional) Select the global elements from which you want to create message definitions. Click **Next**.
9. (Optional) If any errors or warnings are listed, either click **Finish**, if you want the import to be attempted regardless of the errors or warnings listed, or click **Cancel** to terminate the import. You can then correct any errors and attempt the import again.
10. Click **Finish**.

After a WSDL file has been imported into a message set, you can drag the WSDL file onto the message flow editor. The next step is to develop the message flow. For more information about how to start, see "Designing a message flow" on page 177.

Creating an application based on an existing message set

Create a new application that is based on an existing message set.

Before you start:

You must have created a message set by following the instructions in Creating a message set.

To create a new application, complete the following steps.

1. Switch to the Broker Application Development perspective.
2. Open the New Message Broker Application wizard.
   a. Click **File** → **New** → **Project**, or right-click anywhere in the Broker Development view and click **New** → **Project**. The New Project window opens.
   b. Click **Message Brokers**. A list of wizards is displayed.
   c. Select **Start from existing message set** then click **Next**.

   The first panel of the New Message Broker Application wizard is shown.

   The **Start from existing message set** wizard can also be found in the Project section toolbar. Click **Quick Starts** and select the **Start from existing message set** wizard.
3. Set up the basic resources that are required to develop a broker application from an existing message set.
   a. (Optional) If the message set that you want to use is in a compressed (`.zip`) file, click **Import a message set from a .zip file**, and either type the location of the message set in the **.zip file** and **Compressed message set** fields, or click **Browse** and select, and open, the `.zip` file from the list that is displayed. Then select the required message set. If the `.zip` file that you specify does not contain a message set, you receive a message. You can then type a different location for the message set in the **.zip file** field. Otherwise, click **Cancel**.
   b. (Optional) If the message set that you want to use is not in a compressed file, click **Create a new message set by copying an existing message set** and type into the **Message set to copy** field the name of the message set file that you want to copy. A list is displayed of the message set names from which you can choose. Message sets are filtered to show resources in the active working set only.
   c. Click **Next**.

A panel of the New Message Broker Application wizard is shown. In this panel, you can type the names of the projects, the message flow, the message set, and the working set that contains the two new projects.

4. Type into the appropriate fields, the names of the projects, the message flow, the message set, and the working set that contains the two new projects. Default names of the message flow project, the message flow, and the working set are already displayed in the appropriate fields, but you can edit these fields by typing your own names for these resources. However, if the message set is copied from a `.zip` file that is a project interchange file, you cannot edit the names of the message set project and the message set; the names are imported from the `.zip` file.
5. Click **Finish**. The new message set project, message set, message flow project, and message flow are created. A new working set is also created, if required. The new projects are displayed in the specified working set. The contents of the message set project and the message flow project are displayed in the Broker Development view. The message flow is opened in the message flow editor.

The next step is to develop the message flow. For more information about how to start, see "Designing a message flow" on page 177.

Creating an application that uses WebSphere Adapters

Use the **Start from adapter connection** quick start wizard to create an application that uses WebSphere Adapters.

Before you start:

- Before you run the Adapter Connection wizard, you must gather some information from the Enterprise Information System (EIS). For more information about these prerequisite steps, as well as steps to complete after you have run the wizard, see "Connecting to an EIS by using the Adapter Connection wizard" on page 298.

1. Switch to the Broker Application Development perspective.
2. In the Projects section of the Broker Development view, click **Quick Starts**, then click **Start from adapter connection**. The Adapter Connection wizard opens.
3. Follow the on-screen instructions.
4. Click **Finish**.

When you have completed the steps in the wizard, the specified message set project contains a message set with a message type for each business object, and the specified message flow project references the message set project.

Creating an application by using the Configure New Web Service Usage wizard

Use these instructions to generate a message flow by using the Configure New Web Service Usage wizard.

This task topic describes how to create a new application by using the Configure New Web Service Usage wizard.

1. Open a message set project that contains a WSDL file.
2. Select a WSDL file from either the message set or the `ImportFiles` folder and drag the WSDL file onto the Message Flow editor canvas. Validation occurs and if any of the following errors occurs, a message appears.

- The WSDL file does not come from either a message set or `ImportFiles` folder of the message set project.

  For a multiple-file WSDL, the process also checks that either imports inside the main WSDL have been properly imported into the message set, or imports are available in the `ImportFiles` folder.
- The message set that contains the WSDL file does not support any of the SOAP, XMLNSC, XMLNS, or MRM domains.

  However, if the message set that contains the WSDL file does not support only the SOAP domain, you can generate a flow based on the HTTP nodes, and the process continues.
- No HTTP bindings exist in the WSDL file.
- No port types exist in the WSDL file.

For the flow and subflows to be created correctly, the WSDL file that you are dropping onto the Message Flow editor canvas must be WS-I compliant. If no errors occur, the first page of the **Configure New Web Service Usage** wizard appears. For further information about the fields in the wizard, see Configure Web Service Usage details.

3. In Web service usage, select **Expose message flow as a Web service** or **Invoke Web service from message flow**. If you select **Expose message flow as a Web service**, you can use WebSphere Message Broker with other applications on the Web. If you select **Invoke Web service from message flow**, you use WebSphere Message Broker to start the Web service.
4. Select the **Port type** that you are going to use. By default, the initially selected port type is the first one that has at least one HTTP binding associated with it.

   You receive an error message in the following circumstances:
   - The selected port type does not contain at least one operation.
   - No SOAP bindings (with HTTP transport) in the WSDL document are associated with the port type.
5. Select the **Binding** that you are going to use. You receive an error message in the following circumstances:
   - The selected binding has no operations associated with it.
   - The selected binding has no ports associated with it.

   The **Service Port** field lists all the WSDL ports that point to a selected binding.
6. Select the **Binding operations** that you require. By default, only those operations that are implemented by the binding that you choose are selected.

   If you select one of the operations that is not implemented by the selected binding, you receive a warning message, but you can continue.
7. Click **Next**. For further information on the following fields, see File generation details.
8. Select **HTTP nodes** if you have imported the WSDL file from a message set and do not want the default value of **SOAP nodes**. If you select **HTTP nodes**, you see a message that explains the advantages of using the SOAP nodes. By using SOAP nodes, you can use features such as WS_Security and WS_Addressing. However, if the message set does not support the SOAP domain, you receive an error message.

   If you import the WSDL file from the `ImportFiles` folder, you cannot select **SOAP nodes**.

   All the file names that are generated, together with their location, are listed on this page.

A Details pane appears if any warnings occur about the subflow that is generated.

9. Click **Finish** to complete the wizard, create the subflow, and add appropriate nodes to the main flow. For details about the subflow and nodes that generated by the wizard if you select **Expose message flow as a Web service** as the initial step, see "Web service provider message flow generated."

   For details about the subflow and nodes that are generated by the wizard if you select **Invoke Web service from message flow** as the initial step, see "Web service consumer message flow generated" on page 176.

The next step is to develop the message flow. For more information about how to start, see "Designing a message flow" on page 177.

Web service provider message flow generated

This provides additional information in relation to the Configure New Web Service Usage wizard about the message flow generated when the flow is a web service provider.

Note that the default name for the generated subflow is prefixed by the name of the WSDL file you selected.

Generated message flow

The message flow generated consists of a:

SoapInput node
: This `SOAPInput` node fills in the `LocalEnvironment` destination tree with the SOAP operation so that it can be followed either by a:

  - `SOAPExtract` node, or by a
  - `RouteToLabel` node. In this case, appropriate `Label` nodes need to be in place.

  The `out` terminal of the `SOAPInput` node is connected to the `in` terminal of the `SOAPExtract` node.

Subflow node
: The subflow node name reflects the name of the WSDL file.

SOAPReply node
: This node sends the response message back to the originating client.

  Typically, you connect the output of your node, or nodes, that handle your operation, or operations, to the `in` terminal of the `SOAPReply` node.

Generated message subflow

The generated subflow is constructed as follows:

- The input node is connected to the `SOAPExtract` node, which removes the SOAP envelope.

  The `SOAPExtract` node also allows for routing of the SOAP messages, based on the operation being performed. In particular, the SOAP message is routed to a `Label` node within the message flow as identified by the SOAP operation within the message.
- The `Failure` output terminal of the `SOAPExtract` node is connected to the `Output` node used when a process fails named, for example, `failure`.
- A `Label` node is generated for each SOAP operation and each `Label` node is connected to the corresponding `Output` node.

- Each Output node in the subflow corresponds to an output terminal for the SOAPExtract node in the main message flow.

  Therefore, there is one failure output terminal, plus one output terminal for each operation.

  Typically, you connect the output terminal corresponding to the operation you require to the node, or nodes, that handle this operation, for example, Compute node.

Web service consumer message flow generated

This provides additional information in relation to the Configure New Web Service Usage wizard about the message flow generated when the flow is a web service consumer.

Note that the default name for the generated subflow is prefixed by the name of the WSDL file you selected.

Generated message flow

The low generated consists of a single node that has a number of output terminals:

- Failure
- Error
- Fault
- One more, corresponding to the name of the selected operation.

Typically, your message flow feeds an input message to the in terminal of the generated subflow node, and handles various outcomes of the web service invocation.

The default name of the subflow node is a combination of selected operation and WSDL file name. You can change the name of the corresponding .msgflow file on the second page of the wizard; see Configure New Web Service Usage wizard: File generation details.

The generated .msgflow file is placed into the gen folder of the message set project; see "Generated message subflow" for details of this subflow.

Generated message subflow

The generated subflow is constructed as follows:

- A SOAPRequest node immediately follows an Input node. This is a synchronous request and response node that blocks after sending the request, until the response is received. The SOAPRequest node parses the response message.
- The Failure and Error terminals are connected to the Output nodes for failure and error respectively.
- The Out terminal is connected to the SOAPExtract node.

  The SOAPExtract node removes the SOAP envelope so that the body of a SOAP message is extracted.

  The SOAPExtract node also allows for routing of the SOAP messages, based on the operation being performed. Note that only the selected operation and fault are handled.

  In particular, the SOAP message is routed to a Label node within the message flow as identified by the SOAP operation or a ws__Fault label, if fault is returned from the web service.

Each `Label` node is connected to the corresponding `Output` node.

The `Failure` terminal of the `SOAPExtract` node is connected to the `Output` node for failure.

- Each `Output` node in the subflow corresponds to an output terminal for the subflow node.

  Therefore, there are three output terminals:

  – `Failure`

  – `Fault`

  – One for the selected operation.

Designing a message flow

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

Before you start:

Read the following concept topic: “Message flow nodes” on page 6.

When you design a message flow, consider the following questions and options:

- The mode that your broker is working in can affect the types of node that you can use and the number of message flows you can deploy. For more information, see Restrictions that apply in each operation mode.
- Which nodes provide the function that you require. In many cases, you can choose between several nodes that provide a suitable function. You might have to consider other factors listed here to determine which node is best for your overall needs. You can include built-in nodes, user-defined nodes, and subflow nodes. For more information, see “Deciding which nodes to use” on page 179.
- Whether it is appropriate to include more than one input node. For more information, see “Using more than one input node” on page 191.
- How to specify the characteristics of the input message. For more information, see “Defining input message characteristics” on page 192.
- Whether to determine the path that a message follows through the message flow, based on the content or the characteristics of the message. Several nodes provide checks or examination of the message, and have output terminals that can be connected to direct certain messages to different nodes. For more information, see “Using nodes for decision making” on page 193.
- Whether you can use subflows that provide a well-defined subset of processing. You might be able to reuse subflows that were created for another project (for example, an error processing routine), or you might create a subflow in your current project, and reuse it in several places in the same message flow. For more information, see “Using subflows” on page 196.
- What response times your applications expect from the message flow. This factor is influenced by several aspects of how you configure your nodes and the message flow. For more information, see “Optimizing message flow response times” on page 197.
- Whether your message flow processing makes demands on system resources such as stack size. For more information, see “System resources for message flow development” on page 200.
- Whether you can use the destination list in the local environment that is associated with the message to determine the processing in the message flow

(for example, by using RouteToLabel and Label nodes), or the target for the output messages (for example, by setting the Destination Mode property of the MQOutput node to Destination List). For more information, see “Creating destination lists” on page 201.

- Whether to use WebSphere MQ cluster queues. For more information, see “Using WebSphere MQ cluster queues for input and output” on page 202.
- Whether to use WebSphere MQ shared queues on z/OS . For more information, see “Using WebSphere MQ shared queues for input and output (z/OS)” on page 203.
- Whether to validate input messages that are received by the input node, or output messages that are generated by the Compute node, or both. For more information, see “Validating messages” on page 204.
- Whether to view or record message structure in Trace node output. For more information, see “Viewing the logical message tree in trace output” on page 206.
- Whether your message flows access data in databases. You must configure brokers, databases, and database connections to enable this function, as described in Configuring broker and user databases. You must also configure your message flows; see “Accessing databases from message flows” on page 209.

  If you include nodes that use ESQL, for information about how to code the appropriate statements, see “Accessing databases from ESQL” on page 212. If you want to access databases from Java nodes by using JDBC, see “Interacting with databases by using the JavaCompute node” on page 541 or Extending the capability of a Java message processing or output node.

  You can also access databases through the Broker Application Development perspective in the workbench; see “Adding database definitions to the workbench” on page 578.
- Whether your message flows access data in files. By using the FileInput and FileOutput nodes, your message flows can read messages from files and write messages to files in the local file system, or on a network file system that appears local to the broker. For more information, see “Working with files” on page 845.
- Whether your messages must be handled in a transaction. You can set the properties of some built-in nodes to control how transactions are managed, and how messages are processed in a transaction. For more information, see “Configuring transactionality for message flows” on page 213.

  If you want to include JMSInput and JMSOutput nodes in your message flow transactions, you must consider the additional information in “Configuring JMSInput and JMSOutput nodes to support global transactions” on page 215.
- Whether you want your messages to go through data conversion. For information about the available options, see “Configuring message flows for data conversion” on page 224.
- Whether you want to use the MQGet node. For more information about how messages are processed by the MQGet node, and a description of a request-reply scenario that uses this node, see “Using MQGet nodes” on page 226.
- How your message flows can benefit from user exits. For more information, see “Exploiting user exits” on page 239.
- What steps to take to ensure that messages are not lost. For more information, see “Ensuring that messages are not lost” on page 241.
- How errors are handled in the message flow. You can use the facilities provided by the broker to handle any errors that arise during message flow execution (for example, if the input node fails to retrieve an input message, or if writing to a

database results in an error). However, you might prefer to design your message flow to handle errors in a specific way. For more information, see "Handling errors in message flows" on page 244.

- Whether you want a systems monitoring tool to be able to query, discover, and set certain user-defined properties at run time. For more information, see Setting user-defined properties at run time in a CMP application.

For a basic introduction to developing message flows, see the IBM Redbooks publication WebSphere Message Broker Basics. (This link works only if you are connected to the Internet.)

Deciding which nodes to use

WebSphere Message Broker includes many message processing nodes that you can use in your message flows.

Before you start:

Read the concept topic about message flow nodes.

WebSphere Message Broker also provides an interface that you can use to define your own nodes, known as user-defined nodes.

The mode that your broker is working in can affect the types of node that you can use; see Restrictions that apply in each operation mode.

Your decision about which nodes to use depends on the processing that you want to perform on your messages.

Input, output, and request nodes

Input and output nodes define points in the message flow to which client applications send messages (input nodes, such as MQInput), and from which client applications receive messages (output nodes, such as MQOutput). Client applications interact with these nodes by putting messages to, or getting messages from, the I/O resource that is specified by the node as the source or target of the messages. Although a message flow must include at least one input node, it does not need to include an output node.

You can also use reply, request, and response nodes to interact with other applications from within a message flow; these types of node are supplied for a subset of protocols only.

- If you are creating a message flow for deployment to a broker, you must include at least one input node to receive messages. The input node that you select depends on the source of the input messages, and where in the flow you want to receive the messages:

  WebSphere MQ input and get nodes

  Use an MQInput node if the messages arrive at the broker on a WebSphere MQ queue, and the node is to be at the start of a message flow.

  The use of message flows that contain MQeInput nodes in WebSphere Message Broker Version 6.1 is deprecated. Redesign your message flows to remove the MQe nodes and replace them with MQ nodes that are configured to your own specifications and coordinated with your MQe Gateway configuration. For

more details, see Migrating a message flow that contains WebSphere MQ Everyplace® nodes.

Use an MQGet node to retrieve a message from a WebSphere MQ queue, if you want to get the message later in the message flow.

SCADAInput node
Use a SCADAInput node if the messages are sent by a telemetry device.

HTTP input and request nodes
Use an HTTPInput node if the messages are sent by a Web services client.

Use an HTTPRequest node if your message flow interacts with a Web service after it has started.

FileInput node
Use a FileInput node if the messages are contents of files.

TCPIPClientInput or TCPIPServerInput node
Use a TCPIPClientInput node or a TCPIPServerInput node to create a TCP/IP connection when messages are sent through raw TCP/IP sockets.

TCPIPClientReceive or TCPIPServerReceive node
Use a TCPIPClientReceive node or a TCPIPServerReceive node to read the messages that arrive in the message flow through a TCP/IP connection.

Real-timeInput or Real-timeOptimizedFlow node
Use one of these nodes if the messages are sent by a JMS or multicast application.

The Real-timeInput node is an input node; the Real-timeOptimizedFlow node is a complete message flow that provides a high performance publish/subscribe message flow which receives from and sends to JMS or multicast applications.

JMSInput node
Use a JMSInput node if the messages are sent by a JMS application.

WebSphere Adapters nodes
Use the WebSphere Adapters nodes to interact with Enterprise Information Systems (EIS) such as SAP, Siebel, and PeopleSoft. The following input and request nodes are available:
- SAPInput node
- SAPRequest node
- SiebelInput node
- SiebelRequest node
- PeopleSoftInput node
- PeopleSoftRequest node
- TwineballInput node
- TwineballRequest node

The WebSphere Adapters input nodes monitor an EIS for a particular event. When that event occurs, business objects are sent to the input node. The node constructs a tree representation of the business objects and propagates it to the Out terminal so that the data can be used by the rest of the message flow.

The WebSphere Adapters request nodes can send and receive business data. They request information from an EIS and propagate the data to the rest of the message flow.

WebSphere Service Registry and Repository (WSRR) nodes
Use the WebSphere Service Registry and Repository nodes to retrieve Web services information:

- Use the EndpointLookup node to retrieve service endpoint information held in the WebSphere Service Registry and Repository.
- Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository.

SOAP input and request nodes
Use the SOAP nodes to process client SOAP messages and to configure the message flow to behave like a SOAP Web Services provider:

- SOAPAsyncRequest
- SOAPInput
- SOAPRequest

IMSRequest node
Use the IMSRequest node to send a request to run a transaction on a local or remote IBM Information Management System (IMS) system, and wait for a response. IMS Connect must be configured and running on the IMS system.

Input node
If you are creating a message flow that you want to embed in another message flow (a subflow) that you will not deploy as a stand-alone message flow, you must include at least one Input node to receive messages into the subflow.

An instance of the Input node represents an In terminal. For example, if you have included one instance of the Input node, the subflow icon shows one In terminal, which you can connect to other nodes in the main flow in the same way that you connect any other node.

To deploy a message flow, it must have at least one input node. If your message flow does not contain an input node, you are prevented from adding it to the broker archive file. The input node can be in the main flow, or in a message flow that is embedded in the main flow.

You can use more than one input node in a message flow. For more information, see "Using more than one input node" on page 191.

User-defined input node
Use a user-defined input node if the message source is a client or application that uses a different protocol or transport.

- If you want to send the messages that are produced by the message flow to a target application, you can include one or more output nodes. The output node that you select depends on the transport across which the target application expects to receive those messages.

Publication node
Use a Publication node to distribute the messages using the

publish/subscribe network for applications that subscribe to the broker across all supported protocols. A Publication node is an output node that uses output destinations that are identified by subscribers whose subscriptions match the characteristics of the current message.

WebSphere MQ output and reply nodes
Use an MQOutput node if the target application expects to receive messages on a WebSphere MQ queue, or on the WebSphere MQ reply-to queue that is specified in the input message MQMD.

The use of message flows that contain MQeOutput nodes in WebSphere Message Broker Version 6.1 is deprecated. Redesign your message flows to remove the MQe nodes and replace them with MQ nodes that are configured to your own specifications and coordinated with your MQe Gateway configuration. For more details, see Migrating a message flow that contains WebSphere MQ Everyplace nodes.

Use an MQReply node if the target application expects to receive messages on the WebSphere MQ reply-to queue that is specified in the input message MQMD.

SOAP reply and response nodes
Use a SOAPReply node if the target application expects to receive SOAP messages in response to a message sent to the SOAPInput node.

Use a SOAPAsyncResponse node to deliver a SOAP message in response to a message received by the SOAPAsyncRequest node.

SCADAOutput node
Use a SCADAOutput node if a telemetry device is the target of the output messages, and the Publication node is not suitable.

HTTPReply node
Use an HTTPReply node if the messages are in response to a Web services client request.

FileOutput node
Use a FileOutput node if a file is the target of the output messages.

TCPIPClientOutput or TCPIPServerOutput node
Use a TCPIPClientOutput node or a TCPIPServerOutput node if the messages are to be sent to the target application through raw TCP/IP sockets.

JMS output and reply nodes
Use a JMSOutput node if the messages are for a JMS destination.

The JMSReply node has a similar function to the JMSOutput node, but the JMSReply node sends JMS messages only to the reply destination that is supplied in the JMSReplyTo header field of the JMS message tree. Use the JMSReply node to treat a JMS message that is produced from a message flow as a reply to a JMS input message, and when you have no other routing requirements.

EmailOutput node
Use the EmailOutput node to send an e-mail message to one or more recipients.

Output node
If you are creating a message flow that you want to embed in another message flow (a subflow) that you will not deploy as a stand-alone message flow, you must include at least one Output node to propagate messages to subsequent nodes that you connect to the subflow.

An instance of the Output node represents an Out terminal. For example, if you have included two instances of the Output node, the subflow icon shows two Out terminals, which you can connect to other nodes in the main flow in the same way that you connect any other node.

User-defined output node
Use a user-defined output node if the target is a client or application that uses a different protocol or transport.

Nodes for manipulating, enhancing, and transforming messages

Most enterprises have applications that have been developed over many years, on different systems, using different programming languages, and different methods of communication. WebSphere Message Broker removes the need for applications to understand these differences by providing the ability to configure message flows that transform messages from one format to another.

For example, personal names are held in many forms in different applications. Family name first or last, with or without middle initials, uppercase or lowercase: these are just some of the permutations. Because you can configure your message flow to know the requirements of each application, each message can be transformed to the correct format without modifying the sending or receiving application.

You can work with the content of the message to update it in several ways. Your choices here might depend on whether the message flow must handle predefined (modeled) messages, self-defining messages (for example, XML), or both.

A message flow can completely rebuild a message, convert it from one format to another (whether *format* means order of fields, byte order, language, and so on), remove content from the message, or introduce specific data into it. For example, a node can interact with a database to retrieve additional information, or to store a copy of the message (whole or part) in the database for offline processing.

The following examples show how important message transformation can be:

- An order entry application has a Part ID in the body of the message, but its associated stock application expects it in the message header. The message is directed to a message flow that knows the two different formats, and can therefore reformat the information as it is needed.
- A data-entry application creates messages containing stock trade information. Some applications that receive this message need the information as provided, but others need additional information added to the message about the price to earnings (PE) ratio. The stock trade messages are directed to a message flow that passes the message

unchanged to some output nodes, but calculates and adds the extra information for the others. The message flow does this by looking up the current stock price in a database, and uses this value and the trade information in the original message to calculate the PE value before passing on the updated message.

You can also create message flows that use these nodes to interact with each other. Although the default operation of one message flow does not influence the operation of another message flow, you can force this action by configuring your message flows to store and retrieve information in an external source, such as a database.

Compute node

Use the Compute node to:

- Manipulate message content
- Transform the message in some way
- Interact with a database to modify the content of the message or the database and pass on one or more new messages

You can use this node to manipulate predefined and self-defining messages.

Use the ESQL editor to create an ESQL module, specific to this node, that contains the statements that define the actions to perform against the message or database. Do not use the ESQL code that you develop for use in a Compute node in any other type of node.

You can control the way in which the database is accessed by this node by specifying user and password information for the data source that you specify in the node property. Use the mqsisetdbparms command to initialize and maintain these values.

If your message manipulation requirements are complex, perform these in a single Compute node. Fewer, more complex Compute nodes perform better than a larger number of simpler nodes because the broker parses the message on entry to each Compute node.

JavaCompute node

Use the JavaCompute node to:

- Examine an incoming message and, depending on its content, propagate it unchanged to one of the node's output terminals. The node behaves in a similar way to a Filter node, but uses Java instead of ESQL to determine which output terminal to use.
- Change part of an incoming message and propagate the changed message to one of the output terminals.
- Interact with a database through a JDBC type 4 connection to modify the content of the message or the database and pass on one or more new messages
- Create and build a new output message that is totally independent of the input message.

PHPCompute node

Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language. The node functions in a similar way to the JavaCompute node, but uses PHP instead of Java for message transformation and routing.

Mapping node

Use the Mapping node to create a new message from the input message by mapping the content of elements of the output message from elements of the input message, or from database content. You can also extract parts of the message, and optionally change their content, to create a new output message that is a partial copy of the message that is received by the node. The Mapping node handles only predefined messages.

You can control the way in which the database is accessed by this node by specifying user and password information for the data source that you specify in the node property. Use the mqsisetdbparms command to initialize and maintain these values.

Use the Mapping editor to develop mappings to perform simple manipulations on predefined messages. Do not use the mappings that you develop for use in a Mapping node in any other type of node.

Extract node

The Extract node is deprecated in WebSphere Message Broker Version 6.1. Although message flows that contain an Extract node remain valid in WebSphere Message Broker Version 6.1, where possible, redesign your message flows so that any Extract node is replaced by a Mapping node.

With an Extract node you can create a new output message from specified elements of the input message. You can extract parts of the message, and optionally change their content, to create a new output message that is a partial copy of the message received by the node. The Extract node handles only predefined messages.

Use the Mapping editor to develop mappings to perform simple manipulations on predefined messages in the Extract node. Do not use the mappings that you develop for use in an Extract node in any other type of node.

Database node

Use the Database node to interact with a database that is identified by the node properties. The Database node handles both predefined and self-defining messages. Use the ESQL editor to code ESQL functions to update database content from the message, insert new information into the database, and delete information from the database, perhaps based on information in the message. Do not use the ESQL code that you develop for use in a Database node in any other type of node.

This node provides a very flexible interface with a wide range of functions. It also has properties that you can use to control the way in which the interaction participates in transactions.

You can control the way in which the database is accessed by this node by specifying user and password information for the data source that you specify in the node properties. Use the mqsisetdbparms command to initialize and maintain these values.

You can update only databases from this node; you cannot update message content. If you want to update message content, use the Compute or Mapping node.

DataDelete, DataInsert, DataUpdate nodes

The DataDelete, DataInsert, and DataUpdate nodes are specialized forms of the Database node that provide a single mode of interaction (deletion of one or more rows, insertion of one or more rows, or update of one or more existing rows).

The DataDelete, DataInsert, and DataUpdate nodes handle only predefined messages. Use a mapping editor to develop mappings to perform these functions. Do not use the mappings that you develop for these nodes in any other type of node. You can use these nodes to control the transactional characteristics of the updates that they perform.

You can control the way in which the database is accessed by this node by specifying user and password information for the data source that you specify in the node property. Use the mqsisetdbparms command to initialize and maintain these values.

You can update only databases from these nodes; you cannot update message content. If you want to update message content, use the Compute or Mapping node.

Warehouse node

The Warehouse node provides a store interface that you can use to store all or part of the message in a database, for example, for audit reasons. The Warehouse node handles only predefined messages. Use the Mapping editor to develop mappings to perform this action. Do not use the mappings that you develop for a Warehouse node in any other type of node.

You can control the way in which the database is accessed by this node by specifying user and password information for the data source that you specify in the node property. Use the mqsisetdbparms command to initialize and maintain these values.

You can update only a database from this node; you cannot update message content. If you want to update message content, use the Compute or Mapping node.

DatabaseRoute node

Use the DatabaseRoute node to route a message using information from a database in conjunction with applied XPath routing expressions. The node looks up a collection of named column values from a located database row and synchronously applies one or more XPath expressions to these acquired values. Use the DatabaseRoute node to implement message routing with minimal programming logic. For more advanced routing scenarios, use a Compute node or a JavaCompute node.

DatabaseRetrieve node

Use the DatabaseRetrieve node to ensure that information in a message is up to date. Use the node to modify a message using information from a database. For example, you can add information to a message using a key, such as an account number, that is contained in a message. Use the DatabaseRetrieve node to implement message routing with minimal programming logic. For more advanced routing scenarios, use a Compute node or a JavaCompute node.

XSLTransform node

Use the XSLTransform node (formerly known as the XMLTransformation node) to transform an input XML message into another format using XSLT style sheets and to set the message domain, message set, message type, and message format for the generated message. It is imperative that the data can be parsed into a XML message. The style sheet, using the rules that are defined in it, can perform the following actions:

- Sort the data
- Select data elements to include or exclude based on some criteria
- Transform the data into another format

The Xalan-Java transformation engine (Apache Xalan-java XSLT processor) is used as the underlying transformation engine. For more information about XML Transformations, the W3C specification of the syntax, and semantics of the XSL Transformations language for transforming XML documents into other XML documents, see W3C XSL Transformations.

You can deploy style sheets and XML files to broker execution groups, to help with style sheet and XML file maintenance.

JMSMQTransform node

Use the JMSMQTransform node to transform a message with a JMS message tree into a message that has a tree structure that is compatible with the format of messages that are produced by the WebSphere MQ JMS provider.

The JMSMQTransform node can be used to send messages to existing message flows and to interoperate with WebSphere MQ JMS and WebSphere MQ Publish/Subscribe.

MQJMSTransform node

Use the MQJMSTransform node to receive messages that have a WebSphere MQ JMS provider message tree format, and transform them into a format that is compatible with messages that are to be sent to JMS destinations.

You can use the MQJMSTransform node to send messages to existing message flows and to interoperate with WebSphere MQ JMS and WebSphere MQ Publish/Subscribe.

MQOptimizedFlow node

Use the MQOptimizedFlow node to replace a publish/subscribe message flow that consists of an MQInput node connected to a Publication node, and that uses the JMS over WebSphere MQ transport. You cannot include a MQOptimizedFlow node in a message flow that you deploy to z/OS systems.

Use the MQOptimizedFlow node to improve performance, particularly where a single publisher produces a persistent publication for a single subscriber.

SOAPEnvelope and SOAPExtract nodes

Use the SOAPEnvelope and SOAPExtract nodes to add or remove SOAP envelopes from the SOAP message body. You can use the SOAPExtract node both to extract the envelope, and to route the message based on the message content to a Label node.

Header nodes

Use the HTTPHeader, JMSHeader, or MQHeader nodes to

manipulate HTTP, JMS, and WebSphere MQ transport headers and their properties without writing compute nodes. You cannot use these nodes to change the message body.

- Use the HTTPHeader node to add, modify, or delete HTTP headers such as HTTPInput, HTTPResponse, HTTPRequest and HTTPReply.
- Use the JMSHeader node to modify contents of the JMS Header_Values and Application properties so that you can control the node's output without programming.
- Use the MQHeader node to add, modify, or delete MQ Message Descriptor (MQMD) and MQ Dead Letter Header (MQDLH) headers.

User-defined processing node

Use a user-defined node processing node to handle specific requirements that are not met by the built-in nodes.

For example, if your node accesses a database, include a user-defined node to interact with the database. You can control the way in which the database is accessed by this node by specifying user and password information for the data source that you specify in the node property. Use the mqsisetdbparms command to initialize and maintain these values.

Nodes for making decisions

You can use nodes that determine the order and flow of control in the message flow in various ways to make decisions about how messages are processed by the flow. You can also use nodes (TimeoutControl and TimeoutNotification) that determine the time, and frequency of occurrence, of events in the message flow. Routing is independent of message transformation, although the route that a message takes might determine exactly what transformation is performed on it.

For example, a money transfer application always sends messages to one other application. You might decide that every message with a transfer value of more than $10,000 must also be sent to a second application, to enable all high-value transactions to be recorded.

In another example, a national auto club offers a premier service to specific members for orders above a threshold value. Most orders are routed through the typical channels, but, if the membership number and order value meet certain criteria, the order gets special treatment.

You can also establish a more dynamic routing option by building additional routing information into the message when it is processed. Optional sets of rules are set up to receive messages according to values (destinations) set into the message. You can establish these rules such that a message is processed by one or more of the optional sets of rules, in an order determined by the added message content.

Use the following nodes to make decisions about the route that a message follows through the message flow:

Validate node

Use the Validate node to check that the message that arrives on its input terminal is as expected. You can check that the message has the expected message template properties (that is, the message domain, message set and message type) and that the content of the

message is correct. You can check the message against one or more of message domain, message set, or message type.

The Validate node replaces the Check node, which is deprecated in WebSphere Message Broker Version 6.1. The Validate node works in the same way as the Check node, but it has additional Validation properties to enable the validation of message content by parsers that support that capability.

Filter node

Use the Filter node with an ESQL statement to determine the next node to which the message is sent by this node. Do not use the ESQL code that you develop for use in a Filter node in any other type of node.

The node's terminals are True, False, Unknown, and Failure; the message is propagated to the True terminal if the test succeeds, and to the False terminal if it fails. If the statement cannot be resolved (for example, it tests the value of a field that is not in the input message), the message is propagated to the Unknown terminal. If any other error is detected, the message is propagated to the Failure terminal.

The test in the ESQL statement can depend on message content, database content, or a combination of the two.

If you refer to a database, you can control the way in which it is accessed by this node by specifying user and password information for each data source defined in the registry on the broker system. Use the mqsisetdbparms command to initialize and maintain these values.

Use this node in preference to the Compute node to provide message selection and routing; the Filter node is more efficient for this task.

FlowOrder node

You can connect the terminals of this node to force the message to be processed by one sequence of nodes, followed by a second sequence of nodes.

Passthrough node

Use the Passthrough node to enable version control of a subflow at run time. Use this node to add a label to your subflow. By combining this label with a reserved word replacement from your version control system, you can identify which version of a subflow is included in a deployed message flow. You can use this label for your own purposes. If you have included the correct version keywords in the label, you can see the value of the label:

- Stored in the broker archive (BAR) file, using the mqsireadbar command
- As last deployed to a particular broker, on the properties of a deployed message flow in the workbench
- In the broker, if you enable user trace for that message flow

Route node

Use the Route node to direct messages that meet certain criteria down different paths of a message flow. For example, you can forward a message to different service providers, based on the request details. You can also use the Route node to bypass

unnecessary steps. For example, you can check to see if certain data is in a message, and perform a database lookup operation only if the data is missing. If you set the Distribution Mode property to All, you can trigger multiple events that each require different conditions. For example, you can log requests that relate to a particular account identifier, and send requests that relate to a particular product to be audited.

Use the Route node to implement message routing with minimal programming logic. For more advanced routing scenarios, use a Compute node or a JavaCompute node.

RouteToLabel node
Use the RouteToLabel node following a Compute node or a JavaCompute node for complex routing. Define a list of destinations in a Compute or JavaCompute node that are acted on by the RouteToLabel node, which interrogates the destinations and passes the message on to the corresponding Label node.

Label node
Use the Label node as a target for the next sequence of one or more nodes that are to process a message. Use this node in combination with the RouteToLabel node for all types of messages, or with the SOAPExtract node for SOAP messages.

The Label node only routes the message to the next node in the flow and performs no processing.

ResetContentDescriptor node
Use the ResetContentDescriptor node to set new message properties that are used when the message bit stream is next parsed by a subsequent node in the message flow.

Nodes for controlling time-sensitive operations
You might want a batch application process to run every day at a specific time, or you might want information to be processed and published at fixed intervals (for example, currency exchange rates are calculated and sent to banks), or you might want to take a specified recovery action if certain transactions are not completed within a defined time. For all these cases two timeout nodes (TimeoutControl and TimeoutNotification) are provided.

TimeoutControl node
Use a TimeoutControl node and a TimeoutNotification node together in a message flow to control events that occur at a specific time or at defined time intervals. The TimeoutControl node receives an input message that contains a timeout request. All or part of this input message is validated and stored to be propagated by an associated TimeoutNotification node in the message flow. The input message is also propagated unchanged to the next node in the message flow.

More than one TimeoutControl node can be associated with each TimeoutNotification node.

TimeoutNotification node
Use a stand-alone TimeoutNotification node to generate

messages that are propagated at configured times or time intervals to the next node in the message flow for further processing.

Nodes for collating requests

Use the AggregateControl, AggregateReply, and AggregateRequest nodes to collate related requests and responses. Use these nodes to generate several requests in response to one input message, to control and coordinate the responses that are received in response to those requests, and to combine the information that is provided by the responses to continue processing.

Node for creating message collections

Use the Collector node to generate collections of messages and make multiple synchronous or asynchronous requests in parallel. The Collector node does not need an initial fan-out stage, and can group together unrelated input messages by correlating their content. You can configure dynamic input terminals on a Collector node to receive messages from different sources. You can also configure properties on the Collector node, known as event handlers, to determine how messages are added to a message collection, and when a message collection has completed.

Nodes for handling and reporting errors

Use the following nodes to affect error handling and reporting:

Trace node
Include a Trace node to generate one or more trace entries to record what is happening in the message flow at this point.

TryCatch node
Include a TryCatch node to control the error processing when exceptions are thrown.

Throw node
Include a Throw node to force an exception to be thrown, and specify the identity of the exception, to make it easier to diagnose the problem.

Using more than one input node

You can include more than one input node in a single message flow.

Before you start:

Read the following concept topic:

- "Message flow nodes" on page 6

You might find this useful in the following situations:

- The message flow provides common processing for messages that are received from multiple transports. For example, a single message flow might handle:
  - Data in messages received from WebSphere MQ, and therefore through a WebSphere MQ queue and an MQInput node
  - Messages that are received from native IP connections (a Real-timeInput node)
- You need to set standard properties on the MQInput node if input messages:
  - are predefined, and
  - are all received from WebSphere MQ, and

– do not include an MQRFH2 header.

If the required standard properties are not always the same for every message, you can include more than one input node and configure each to handle a particular set of properties.

This requirement is not necessary for self-defining messages.

- Each input node in a message flow causes the broker to start a separate thread of execution. Including more than one input node might improve the message flow performance. However, if you include multiple input nodes that access the same input source (for example, a WebSphere MQ queue), the order in which the messages are processed cannot be guaranteed. If you want the message flow to process messages in the order in which they are received, this option is not appropriate.

  If you are not concerned about message order, consider using additional instances of the same message flow rather than multiple input nodes. If you set the Additional Instances property of the message flow when you deploy it to the broker, multiple copies of the message flow are started in the execution group. This is the most efficient way of handling multiple instances.

Look at the following sample :

- Scribble

This sample uses two input nodes: an MQInput node and a Real-timeInput node. You can use these two input nodes to enable the sample's message flow to accept input from both WebSphere MQ transport and native IP connections. You can view samples information only when you use the information center that is integrated with the Message Broker Toolkit or the online information center.

Defining input message characteristics

When a message is received by an input node in a message flow, the node detects how to interpret that message by determining the domain in which the message is defined and starting the appropriate parser.

Before you start:

Read the following concept topic:

- "Parsers" on page 91

You can provide message domain information to the input node in one of two ways:

1. You can configure the built-in input nodes to indicate the message domain, and therefore the parser to be started, for each message that is received.
2. You can set values in the input message itself that specify this information. Include an MQRFH2 header, which contains a folder that defines the message characteristics. This approach is more flexible because it means that the input node can start the appropriate parser based on the content of each message.

If the input message is defined in the MRM domain, and is therefore interpreted by the MRM parser, you must specify the following additional properties:

- The Message set within which the message is defined
- The Message type, which is defined by the message model
- The Message format, which defines the physical characteristics of the message

The way that these properties are set depends upon the type of message, or node, that you want to use:

- If the message is a WebSphere MQ message, these properties can be set either in the input node or in the MQRFH2 header of the incoming message. If the properties are set in both, the properties of the MQRFH2 header take precedence. If the properties are not found in either the node or the MQRFH2 header, the default value is empty and the BLOB parser is used.
- If the message is a JMS message, the property that is set on the node takes precedence. If the Message domain is empty, the Message domain is, by default, derived according to certain criteria following a predetermined order of precedence; see JMS message payload and appropriate parser.
- If the input message belongs to a Message domain other than those for which a parser is supplied, you must provide a user-defined parser to handle it, and a user-defined input node to accept it for processing in the message flow. Check the documentation provided with the user-defined parser and node for further information.
- If the Message domain is in a TimeoutControl node, an empty Message domain has either of the following results:
  - If the Stored message location property is also empty, the full message is stored. When the message comes back at TimeoutNotification, it is parsed in the same way as the original message.
  - If the Stored message location property is not empty, a partial message is stored and no parser is associated, therefore, by default, it is treated as BLOB.
- If the Message domain is in a ResetContentDescriptor node, an empty Message domain has either of the following results:
  - If Reset message domain is cleared, the domain is not reset.
  - If Reset message domain is selected, the default is BLOB.
- If the input node cannot determine the message characteristics, the default value is empty and the message is considered to be in the BLOB domain, and the BLOB parser is started.

Import either of the following samples, or another sample that uses a Message set, from the Samples Gallery, and look at the values on the **Input Message Parsing** properties tab of the input node in the sample's message flow.

- Video Rental
- Comma Separated Value (CSV)

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

Using nodes for decision making

You can use several built-in nodes in different ways to control the path that a message takes through the message flow.

Before you start:

Read the concept topic about message flow nodes.

These nodes let you decide how messages are processed by specifying the route that each message takes through the message flow based on dynamic values such as message structure and content.

For more information, see the following topics:

- “Testing the message structure (Validate node)” on page 194
- “Controlling the order of processing in a message flow” on page 194

- "Using the destination list to route messages (RouteToLabel and Label nodes)" on page 195

You can use one of the following nodes to determine the path taken by a message through the message flow based on its content:

- "Route node" on page 1142
- Filter node (ESQL)
- JavaCompute node
- PHPCompute node

Use the DatabaseRoute node to route messages using information from a database. For more information, see "DatabaseRoute node" on page 915.

Testing the message structure (Validate node)

Use the Validate node to test the characteristics of the message structure.

If you set the Validate node properties appropriately, you can request that one or all of the message domain, message set, and message type are compared to a specific value. If the message matches those values for which you have requested the check, it is routed through the match terminal and is processed by the sequence of nodes that you have connected to that terminal.

If the message does not match any one of those values for which you have requested the check, it is routed through the failure terminal and is processed by the sequence of nodes that you have connected to that terminal.

For example, you might design a message flow that provides additional processing for all messages that are in the MRM domain. You can include a Validate node that tests just that characteristic of the message, and passes it to a sequence of nodes that provide the specialized processing. If the message is not in the MRM domain, the extra nodes are bypassed, and the failure terminal is wired up directly to the node that follows the sequence required for MRM messages only.

Controlling the order of processing in a message flow

Use the FlowOrder node to control the order of processing in a message flow.

When you connect message flow nodes together, the broker determines the way in which the different connections are processed. This includes the order in which they are processed. If you have connected more than one node or sequence of nodes to a single output terminal, you cannot predict whether one sequence is processed before another for a message.

If the order of processing is important in your message flow, use the FlowOrder node to force a prescribed order of processing of the messages that are propagated by this node.

The FlowOrder node has two output terminals that you can connect to control the order in which subsequent nodes process the message. The output terminals, named First and Second, are always processed in that order.

When you connect a node or sequence of nodes to the First terminal, the input message is passed to the next node, and all processing defined by all subsequent nodes in this sequence is completed before control returns to the FlowOrder node.

The input message is then propagated to the next node in the sequence of nodes connected to the Second terminal.

The message passed to both sequences of nodes, from the First terminal and the Second terminal, is identical. It is always the message that the FlowOrder node receives as input. The message that the FlowOrder node propagates to the Second terminal is in no way affected by the processing of the message that has been performed by the sequence of nodes connected to the First terminal.

The FlowOrder node provides no other processing on the input message; it is used only for imposing order on subsequent processing.

Testing the message content (Filter node)

You can use the Filter node to determine the path taken by a message through the message flow based on its content.

You can customize the Filter node by using ESQL statements to determine if the message content meets some condition. The condition tested must yield a Boolean result, that is it must be true or false (or unknown). You can create the test to reference information from a database, if applicable.

You can connect nodes following the Filter node to the corresponding terminals of the Filter node, and process the message according to its content.

Look at the following samples to see how to use the Filter node:
- Airline Reservations
- Error Handler

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

Using the destination list to route messages (RouteToLabel and Label nodes)

You can determine the path that a message takes through the message flow by using the RouteToLabel and Label nodes.

These nodes provide a more flexible way to process messages than the Filter node, which depends on the Boolean result of an ESQL expression for its logic.

When you use RouteToLabel and Label nodes, you must include a Compute node that determines, by using some combination of message content, database content, and ESQL logic, how messages are to be processed next. Configure the Compute node to create a destination list (in the DestinationList folder in the local environment subtree) that contains the destination for each message, specified as the LabelName of a Label node. The Compute node passes the message to the RouteToLabel node, which reads the destination list and propagates the message to either the first or last item on the destination list, according to the value that is specified for the RouteToLabel node's Mode property. Although there is no limit to the number of destinations that the Compute node writes in the destination list, the RouteToLabel node propagates the message only to a single label node. This use of the destination list is in contrast to its use to define the final recipients of the output messages. For more information about the procedure for creating a destination list, see “Creating destination lists” on page 201.

If you intend to derive destination values from the message itself, or from a database, you might also need to cast values from one type to another. For more

information about the local environment, see “Local environment tree structure” on page 80. For more information about casting, see Supported casts.

Look at the following sample to see how to use these nodes:
- Airline Reservations

The XML_PassengerQuery message flow in the previous sample demonstrates how you can use the destination list in the local environment to route messages based on the information in the message itself.

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

Using subflows

You can include subflows in your message flows in exactly the same way as you include built-in or user-defined nodes. You can also connect subflows to other nodes in the same way.

You can define a subflow once, and use it in more than one message flow, and in more than one message flow project. Therefore, a subflow provides the following benefits:
- Reusability and reduced development time.
- Consistency and increased maintainability of your message flows (consider a subflow as analogous to a programming macro, or to inline code that is written once but used in many places).
- Flexibility to tailor a subflow to a specific context (for example, by updating the output queue or data source information).

However, remember that a subflow is not a single node, and its inclusion increases the number of nodes in the message flow, which might affect its performance.

Consider these examples of subflow use:
- You can define a subflow that provides a common sequence of actions that applies to several message flows if an error is encountered; for example, you might have a common error routine that writes the message to a database through the Warehouse node, and puts it to a queue for processing by an error recovery routine. The use of this routine in multiple message flows, or in several places within one message flow, provides an efficient and consistent use of resources and avoids reinventing such routines every time an error is encountered.
- You might want to perform a common calculation on messages that pass through several different message flows; for example, you might access currency exchange rates from a database and apply these to calculate prices in several different currencies. You can include the currency calculator subflow in each of the message flows in which it is appropriate.

Use the Passthrough node to enable version control of a subflow at run time. By including a Passthrough node, you can add a label to your message flow or subflow. By combining this label with keyword replacement from your version control system, you can identify which version of a subflow is included in a deployed message flow. You can use this label for your own purposes. If you have included the correct version keywords in the label, you can see the value of the label:
- Stored in the broker archive (BAR) file, by using the mqsireadbar command

- As last deployed to a particular broker, on the properties of a deployed message flow in the Message Broker Toolkit
- In the runtime environment, if you enable user trace for that message flow

The message that it propagates on its Out terminal is the same message that it received on its In terminal; for example, if you develop an error processing subflow to include in several message flows, you might want to modify that subflow. However, you might want to introduce the modified version initially to just a subset of the message flows in which it is included. Set a value for the instance of the Passthrough node that identifies which version of the subflow you have included.

The use of subflows is demonstrated in the following samples:

- Error Handler
- Coordinated Request Reply

Error Handler uses a subflow to trap information about errors and store the information in a database. Coordinated Request Reply uses a subflow to encapsulate the storage of the ReplyToQ and ReplyToQMgr values in a WebSphere MQ message so that the processing logic can be reused in other message flows, and to allow alternative implementations to be substituted.

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

Adding keywords to subflows

You can embed keywords in each subflow that you use in a message flow.

You must use a different keyword in each instance of a subflow, because only the first recorded instance of each keyword within the message flow `.cmf` file is available to applications that use the Configuration Manager Proxy, which include the Message Broker Toolkit.

The order that subflows appear in the `.cmf` file is not guaranteed.

Optimizing message flow response times

You can use different solutions to improve message flow response times.

Before you start:

Read the following concept topic:

- "Message flow nodes" on page 6

When you design a message flow, the flexibility and functional capabilities of the built-in nodes often mean that there are several ways to achieve the processing and results that you require. You might find that different solutions deliver different levels of performance and, if performance is an important consideration for you, take it into account when designing your message flow

Your applications can perceive performance in either of these ways:

- The response time indicates how quickly each message is processed by the message flow. The response time is particularly influenced by how you design your message flows. Response time is discussed in this topic.
- The throughput indicates how many messages of particular sizes can be processed by a message flow in a specified time. The throughput is mainly

affected by configuration and system resource factors, and is discussed in Optimizing message flow throughput, with other domain configuration information.

Several aspects influence message flow response times. However, as you create and modify your message flow design to arrive at the best results for your specific business requirements, also consider the eventual complexity of the message flow. The most efficient message flows are not necessarily the easiest to understand and maintain; experiment with the solutions available to arrive at the best balance for your needs.

Several factors influence message flow response times:

The number of nodes that you include in the message flow

Every node increases the amount of processing required in the broker, therefore, consider the content of the message flow carefully, including the use of subflows.

Use as few nodes as possible in a message flow; every node that you include in the message flow increases the amount of processing required in the broker. The number of nodes in a single flow has an upper limit, which is governed by system resources, particularly the stack size. For more information about stack sizes, see "System resources for message flow development" on page 200.

How the message flow routes and processes messages

In some situations, you might find that the built-in nodes, and perhaps other nodes that are available in your system, provide more than one way of providing the same function. Choose the simplest configuration. For example, to define some specific processing based on the value of a field in each message, you might design a message flow that has a sequence of Filter nodes to handle each case. If appropriate, you can group messages through the Filter node to reduce the number of nodes through which each message type has to pass before being processed.

For example, you might have a message flow that handles eight different messages (Invoice, Despatch Note, and so on). You can include a Filter node to identify each type of message and route it according to its type. You can optimize the performance of this technique by testing for the most frequent message types in the earliest Filter nodes. However, the eighth message type must always pass through eight Filter nodes.

If you can group the message types (for example, if the message type is numeric, and you can test for all types greater than four and not greater than four), you can reduce the number of Filter nodes required. The first Filter node tests for greater than four, and passes the message on to two further Filter nodes (attached to the True and False terminals) that test for less than two and less than six. An additional four Filter nodes can then test for one or two, three or four, and so on. Although the number of Filter nodes required is the same, the number of nodes through which each message passes is reduced.

You might find that using a RouteToLabel node with a set of Label nodes provides a better alternative to a sequence of Filter nodes. Each message passes through a smaller number of nodes, improving throughput. However, you must also consider that using a RouteToLabel node necessitates the use of a Compute node: the increase in the amount of processing required in the broker that is caused by the node might

outweigh the advantages. If you are dealing with a limited number of message types, a small number of Filter nodes is more efficient.

The following sample demonstrates how you can use the RouteToLabel and Label nodes instead of using multiple Filter nodes in the XML_PassengerQuery message flow.

- Airline Reservations

The following sample demonstrates how you can store routing information in a database table in an in-memory cache in the message flow.

- Message Routing

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

If your message flow includes loops

Avoid loops of repeating nodes, which can be very inefficient and can cause performance and stack problems. You might find that a Compute node with multiple PROPAGATE statements avoids the need to loop around a series of nodes.

The efficiency of the ESQL

Check all the ESQL code that you have created for your message flow nodes. As you develop and test a node, you might maintain statements that are not required when you have finalized your message processing. You might also find that something you have coded as two statements can be coded as one. Taking the time to review and check your ESQL code might provide simplification and performance improvements.

If you have imported message flows from a previous release, check your ESQL statements against the ESQL available in Version 6.1 to see if you can use new functions or statements to improve its efficiency.

The use of persistent and transactional messages

Persistent messages are saved to disk during message flow processing. You can avoid this situation by specifying that messages are non-persistent on input, output, or both. If your message flow is handling only non-persistent messages, check the configuration of the nodes and the message flow itself; if your messages are non-persistent, transactional support might be unnecessary. The default configuration of some nodes enforces transactionality; if you update these properties and redeploy the message flow, response times might improve.

Message size

A larger message takes longer to process. If you can split large messages into smaller units of information, you might be able to improve the speed at which they are handled by the message flow. The following sample demonstrates how to minimize the virtual storage requirements for the message flow to improve a message flow's performance when processing potentially large messages.

- Large Messaging

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

Message format

Although WebSphere Message Broker supports multiple message formats, and provides facilities that you can use to transform from one format to

another, this transformation increases the amount of processing required in the broker. Make sure that you do not perform any unnecessary conversions or transformations.

You can find more information about improving the performance of a message flow in a developerWorks article (developerWorks article on message flow performance).

System resources for message flow development

Configure your message flows to make the best use of computer resources, especially if you will process large messages.

As well as designing your message flow to optimize throughput, you must ensure that particular areas of storage are efficiently used so that your system does not suffer from capacity issues, and that processes do not end because of lack of resources.

Consider the following storage issues when developing your message flows:

- "Stack storage"
- "JVM heap sizing" on page 201

Stack storage

Depending on the design of your message flow, you might need to increase the stack size.

When a message flow thread starts, it requires storage to perform the instructions that are defined by the message flow nodes. This storage comes from the execution group's heap and stack size. The default stack size that is allocated to a message flow thread depends on the operating system that is used:

Windows On Windows, each message flow thread is allocated 1 MB of stack space.

Linux On Linux, each message flow thread is allocated 8 MB of stack space.

UNIX On UNIX, each message flow thread is allocated 1 MB of stack space.

z/OS On z/OS, each message flow thread is allocated 512 KB of downward stack space and 50 KB of upward stack space.

In a message flow, a node typically uses 2 KB of the stack space. A typical message flow can therefore include 250 nodes on z/OS, 500 nodes on UNIX systems and 500 nodes on Windows. This amount can be higher or lower depending on the type of nodes used and the processing that they perform.

In WebSphere Message Broker, any processing that involves nested or recursive processing can cause extensive usage of the stack. For example, in the following situations you might need to increase the stack size:

- When a message flow is processing a message that contains a large number of repetitions or complex nesting.
- When a message flow is executing ESQL that calls the same procedure or function recursively, or when an operator (for example, the concatenation operator) is used repeatedly in an ESQL statement.

You can increase the stack size to improve performance. For details, see:

- Increasing the stack size on Windows, Linux, and UNIX systems
- Increasing the stack size on z/OS

JVM heap sizing

The Java virtual machine (JVM) heap is an independent memory allocation that can reduce the capacity of the main memory heap.

Every execution group creates its own JVM. The execution group uses the JVM to execute the internal administration threads that require Java. This usage is typically minimal. The primary use of the JVM is for the message flow nodes that include the IBM primitives, user defined extensions, and Java routines that are called from ESQL. You cannot control how much of the JVM heap the IBM primitives use, but you can affect usage of the JVM heap in the Java that is implemented in the following resources:

- A Java user-defined extension node
- A JavaCompute node
- A Java routine that is called from ESQL

From WebSphere Message Broker Version 6.1 onwards, the JVM is created with a minimum of 32 MB of space, and a maximum of 256 MB, allocated and reserved for its use. As with any JVM, you can pass parameters in to set the minimum and maximum heap sizes. Note that on 32-bit platforms, the JVM reserves heap space based on the maximum heap size.

You might need to increase the maximum heap size allocated if you plan to run large messages through the Java primitive nodes listed above.

To give more capacity to a message flow that is going to process large messages, reduce the minimum JVM heap size to allow the main memory heap to occupy more address space. For details of how to reduce the minimum JVM heap size, see Setting the JVM heap size.

Creating destination lists

Create a list of destinations to indicate where a message is sent.

Before you start:

Read the concept topic “Message flow nodes” on page 6.

You can include a Compute node in your message flow, and configure it to create a destination list in the local environment subtree. You can then use the destination list in the following nodes:

- The MQOutput and JMSOutput nodes, to put output messages to a specified list of destinations.
- The RouteToLabel node, to pass messages to Label nodes.

  For details about how this technique is used, look at the following sample:

  – Airline Reservations

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

For more information about accessing the LocalEnvironment subtree, destination list contents, and example procedures for setting values for each of these scenarios, see “Accessing the local environment tree” on page 353.

For more information about how to populate destination in the LocalEnvironment subtree, and how to build JMS destination lists, see “Populating Destination in the local environment tree” on page 355.

You might find it useful to create the contents of the destination list from an external database that is accessed by the Compute node. You can then update the destinations without needing to update and redeploy the message flow.

The use of the destination list to define which applications receive the output messages is in contrast to the publish/subscribe application model, in which the recipients of the publications are those subscribers that are currently registered with the broker. The processing that is completed by the message flow does not have any effect on the current list of subscribers.

Using WebSphere MQ cluster queues for input and output

Design your broker domain to use WebSphere MQ queues, if appropriate for your business needs.

The use of queue manager clusters has the following significant benefits:

1. Reduced system administration

   Clusters need fewer definitions to establish a network; you can set up and change your network more quickly and easily.
2. Increased availability and workload balancing

   You can benefit by defining instances of the same queue to more than one queue manager, therefore distributing the workload through the cluster.

If you use clusters with WebSphere Message Broker, consider the following queues:

For SYSTEM.BROKER queues:
The SYSTEM.BROKER queues are defined for you when you create WebSphere Message Broker components, and are not defined as cluster queues. Do not change this attribute.

For broker, Configuration Manager, and User Name Server connectivity:
If you define the queue managers that support your brokers, the Configuration Manager, and the User Name Server to a cluster, you can benefit from the simplified administration provided by WebSphere MQ clusters. You might find this option relevant for the brokers in a collective, which must all have WebSphere MQ interconnections.

For message flow input queues:
If you define an input queue as a cluster queue, consider the implications for the order of messages or the segments of a segmented message. The implications are the same as for any WebSphere MQ cluster queue. In particular, the application must ensure that, if it is sending segmented messages, all segments are processed by the same target queue, and therefore by the same instance of the message flow at the same broker.

For message flow output queues:

- WebSphere Message Broker always specifies MQOO_BIND_AS_Q_DEF when it opens a queue for output. If you expect segmented messages to be put to an output queue, or want a series of messages to be handled by the same process, you must specify DEFBIND(OPEN) when you define that queue. This option ensures that all segments of a single message, or all messages in a sequence, are put to the same target queue and are processed by the same instance of the receiving application.

- If you create your own output nodes, specify MQOO_BIND_AS_Q_DEF when you open the output queue, and DEFBIND(OPEN) when you define the queue, if you need to ensure message order, or to ensure a single target for segmented messages.

For publish/subscribe applications:

- If the target queue for a publication is a cluster queue, you must deploy the publish/subscribe message flow to all the brokers on queue managers in the cluster. However, the cluster does not provide any of the failover function to the broker domain topology and function. If a broker to which a message is published, or a subscriber registers, is unavailable, the distribution of the publication or registration is not taken over by another broker.
- When a client registers a subscription with a broker that is running on a queue manager that is a member of a cluster, the broker forwards a proxy registration to its neighbors in the broker domain; the registration details are not advertised to other members of the cluster.
- A client might choose to become a clustered subscriber, so that its subscriber queue is one of a set of clustered queues that receive a particular publication. In this case, when registering a subscription, use the name of an "imaginary" queue manager that is associated with the cluster; this queue manager is not the one to which the publication is sent, but an alias for the broker to use. As an administrative activity, a blank queue manager alias definition is made for this queue manager on the broker that satisfies this subscription for all clustered subscribers. When the broker publishes to a subscriber queue that names this queue manager, resolution of the queue manager name results in the publication being sent to any queue manager that hosts the subscriber cluster queue, and only one clustered subscriber receives the publication.

  For example, if the clustered subscriber queue was SUBS_QUEUE and the "imaginary" subscriber queue manager was CLUSTER_QM, the broker definition is:

  ```
  DEFINE QREMOTE(CLUSTER_QM) RQMNAME(' ') RNAME(' ')
  ```

  This configuration sends broker publications for SUBS_QUEUE on CLUSTER_QM to one instance of the cluster queue named SUBS_QUEUE anywhere in the cluster.

To understand more about clusters, and the implications of using cluster queues, see the *Queue Manager Clusters* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.

Using WebSphere MQ shared queues for input and output (z/OS)

On z/OS systems, you can define WebSphere MQ shared queues as input and output queues for message flows.

Use the WebSphere MQ for z/OS product facilities to define these queues and specify that they are shared.

For more information about configuring on z/OS, refer to the *z/OS Concepts and Planning* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.

If you use shared queues, you can provide failover support between different images running WebSphere Message Broker on a sysplex.

You cannot use shared queues for broker or User Name Server component queues such as the SYSTEM.BROKER.CONTROL.QUEUE.

Shared queues are available only on z/OS.

Validating messages

The broker provides validation based on the message set for predefined messages.

Before you start:

Read the concept topics about message flows and parsers, especially “MRM parser and domain” on page 120 and “XMLNSC parser” on page 104.

Validation applies only to messages that you have modeled and deployed to the broker. Specifically, the message domains that support validation are MRM, XMLNSC, SOAP, and IDOC.

The broker does not provide any validation for self-defining messages. The MRM and IDOC parsers validate predefined messages against the message dictionary generated from a message set. The XMLNSC and SOAP domains validate predefined messages directly against XML Schema generated from a message set.

Message flows are designed to transform and route messages that conform to certain rules. By default, parsers perform some validity checking on a message, but only to ensure the integrity of the parsing operation. However, you can validate a message more stringently against the message model contained in the message set by specifying validation options on certain nodes in your message flow.

You can use validation options to validate the following messages:
- Input messages that are received by an input node
- Output messages that are created, for example, by a Compute, Mapping, or JavaCompute node

These validation options can ensure the validity of data entering and leaving the message flow. The options provide you with some degree of control over the validation performed to:
- Maintain a balance between performance requirements and security requirements
- Validate at different stages of message flow completion; for example, on input of a message, before a message is propagated, or at any point in between
- Cope with messages that your message model does not fully describe

You can also specify what action to take when validation fails.

Message validation involves navigating a message tree, and checking the validity of the tree. Message validation is an extension of tree creation when the input message is parsed, and of bit stream creation when the output message is written.

Validation options are available on the following nodes:

Node type	Nodes with validation options
Input node	FileInput, HTTPInput, JMSInput, MQInput, SCADAInput, SOAPInput,TimeoutNotification,
Output node	FileOutput, HTTPReply, JMSOutput, JMSReply, MQOutput, MQReply, SCADAOutput, SOAPReply
Other nodes	Compute, , DatabaseRetrieve, HTTPRequest, JavaCompute, Mapping, MQGet, ResetContentDescriptor, SOAPRequest, SOAPAsyncResponse, Validate, XSLTransform

Validation options can also be specified on the ESQL CREATE statement and the ASBITSTREAM function.

To validate input messages that are received on an input node, you can specify validation properties on the input node. The input message is then validated when the message bit stream is parsed to form the message tree.

You can also use the Parse Timing property of the input node to control whether the entire message is parsed and validated at this time, or whether individual fields in the message are parsed and validated only when referenced.

To validate output messages that are created by a transformation node, specify validation properties either on the node itself, or on the output node that sends the message. The validation takes place when the message bit stream is created from the message tree by the output node.

Alternatively, use a Validate node to validate a message tree at a particular place in your message flow, or use the ESQL ASBITSTREAM function in a Compute, Filter, or Database node.

A limited amount of validation occurs by default if you leave the validation settings unaltered. At this default level, an exception is thrown if one of the following statements is true:

- A data mismatch occurs; for example, the parser cannot interpret the data that is provided for the field type specified.
- The order of elements in the output message does not match the order of elements in the logical message tree (MRM, CWF, and TDS fixed length models only).

Additionally, the MRM parser performs limited remedial action under the following circumstances:

- Extraneous fields are discarded on output for fixed formats (CWF and TDS fixed length models only).
- If mandatory content is missing, default values are supplied, if available, on output for fixed formats (CWF and TDS fixed length models only).
- If an element's data type in the tree does not match that specified in the dictionary, the data type is converted on output to match the dictionary definition, if possible, for all formats.

However, by using validation options you can request more thorough validation of messages. For example, you might want to validate one or more of the following conditions, and throw an exception, or log the errors:

- The whole message at the start of the message flow

- That complex elements have the correct Composition and Content Validation
- That all data fields contain the correct type of data
- That data fields conform to the value constraints in the message model
- That all mandatory fields are present in the message
- That only the expected fields are present in the message
- That message elements are in the correct order

The samples in the Samples Gallery illustrate some of these validation options.

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

When using validation options, it is important to understand the following behavior.

- The Parse Timing property, which controls whether *on-demand* parsing (sometimes called partial parsing) takes place, affects the timing of the validation of input messages, including message headers.

  For more information about the Parse Timing property, see "Parsing on demand" on page 1449.
- If a message tree is passed to an output node, by default the output node inherits the validation options in force for the message tree. You can override these options by specifying a new set of validation options on the output node.
- If a message tree is passed as input to a Compute, Mapping, XSLTransform, DatabaseRetrieve, or JavaCompute node, any new output message trees that the node creates have the validation options specified by the node itself (even if the whole message is copied). You can override this behavior and specify that the messages that are created by the node inherit the validation options of the input message tree.
- (MRM domain only) When the bit stream is written, and validation options are applied, the entire message is validated. The message tree might contain an unresolved type (for example, if a Compute node copied an unresolved type from an input message to an output message without resolving it). If such a type is encountered, a validation error occurs because it is not possible to validate the type. To prevent this error, ensure that all unresolved types are resolved before they are copied to output messages.
- (MRM domain only) Do not select the Truncate fixed length strings check box because validation is done before truncation, and a fixed length field fails validation if its length exceeds the length that is defined in the message set. For more information about the Truncate fixed length strings property, see Custom Wire Format message set properties and TDS Format message set properties.

For information about how you can control validation by using different properties, see "Validation properties" on page 1445.

Viewing the logical message tree in trace output

To view the structure of the logical message tree at any point in the message flow, include a Trace node and write some or all the message (including headers and all four message trees) to the trace output destination.

You might find trace output useful to check or record the content of a message before and after a node has changed it, or on its receipt by the input node. For example, if you include a Compute node that builds a destination list in the local

environment tree, you might want a record of the structure that it has created as part of an audit trail, or you might want to check that the Compute node is working as you expect it to.

UNIX On UNIX, syslog entries are restricted in length and messages that are sent to the syslog are truncated by the newline character. To record a large amount of data in a log on UNIX, set the Destination property on the Trace node to File or User Trace instead of Local Error Log.

1. Switch to the Broker Application Development perspective.
2. Open the message flow for which you want to view messages. Open an existing message flow, or create a message flow.
3. Include a Trace node wherever you want to view part or all the message tree structure. You can include as many Trace nodes as you choose; however, each node that you introduce can affect the performance of message flow processing.
4. Set the Trace node properties to trace the message, or parts of the message, that you want to view. Specify the parts of the message by using ESQL field references. Several examples are included later in this topic.
5. If you have added a Trace node to investigate a particular behavior of your message flow, and have now resolved your concerns or checked that the message flow is working correctly, remove the Trace node or nodes, and redeploy the message flow.

Assume that you have configured a message flow that receives an XML message on a WebSphere MQ queue in an MQInput node. The input message includes an MQRFH2 header. The message has the following content:

```
<Trade type='buy'
 Company='IBM'
 Price='200.20'
 Date='2000-01-01'
 Quantity='1000'/>
```

You can include and configure a Trace node to produce output that shows one or more of the trees created from this message: the message body, environment, local environment, and exception trees. If you choose to record the content of the message body, the Properties tree and the contents of all headers (in this example, at least an MQMD and an MQRFH2) are included. You specify what you want to be recorded when you set the Trace node property Pattern. You can use most of the correlation names to define this pattern (you cannot use those names that are specific to the Compute node).

Message body

If you want the Trace node to write the message body tree including Properties and all headers, set Pattern to $Root. If you want only the message data, set Pattern to ${Body}.

The trace output generated for the message tree of the preceding message with Pattern set to $Root would look like the following example:

```
Root
      Properties
         CreationTime=GMTTIMESTAMP '1999-11-24 13:10:00'          (a GMT timestamp field)

  ... and other fields ...

      MQMD
         PutDate=DATE '19991124'                                  (a date field)

         PutTime=GMTTIME '131000'                                 (a GMTTIME field)

  ... and other fields ...

      MQRFH
          mcd
          msd='xml'                                              (a character string field)

         .. and other fields ...

      XML
         Trade
          type='buy'                                             (a character string field)

         Company='IBM'                                           (a character string field)

         Price='200'                                             (a character string field)

         Date='2000-01-01'                                       (a character string field)

         Quantity='1000'                                         (a character string field)
```

Environment

To trace any data in the environment tree, set Pattern to ${Environment}. This setting produces output like the following example:

```
(0x1000000)Environment = (
     (0x1000000)Variables = (
       (0x1000000)MyVariable1 = (
         (0x2000000) = '3'
       )
       (0x1000000)MyVariable2 = (
         (0x2000000) = 'Hello'
       )
     )
 )
```

To trace particular variables in the variables folder of the environment tree, you can use a more specific pattern, for example ${Environment.Variables.MyVariable1}. This setting returns the value only (for example, it returns just the value 3).

LocalEnvironment

To trace data in the local environment tree, set Pattern to ${LocalEnvironment}. The output you get is like the following example, which shows that a destination list has been created in the local environment tree:

```
(0x1000000)Destination = (
    (0x1000000)MQ                = (
      (0x1000000)DestinationData = (
        (0x3000000)queuename = 'MQOUT'
      )
    )
     (0x1000000)MQDestinationList = (
      (0x1000000)DestinationData = (
        (0x3000000)queuename = 'OLDMQOUT'
      )
    )
    (0x1000000)RouterList        = (
      (0x1000000)DestinationData = (
        (0x3000000)labelname = 'continue'
      )
      (0x1000000)DestinationData = (
        (0x3000000)labelname = 'custdetails'
      )
      (0x1000000)DestinationData = (
        (0x3000000)labelname = 'trade'
      )
    )
  )
```

Another example, shown here, includes a WrittenDestination folder. This example represents a trace that has been written by a Trace node that is included after an MQOutput node, where the Out terminal of the MQOutput node is connected to a sequence of nodes including the Trace node. When an Out terminal is connected, the local environment is augmented with information about the action that the output node has performed.

```
(0x1000000)Destination        = (
    (0x1000000)MQ                = (
      (0x1000000)DestinationData = (
        (0x3000000)queuename = 'MQOUT'
      )
    )
  (0x1000000)WrittenDestination = (
    (0x1000000)MQ = (
      (0x1000000)DestinationData = (
        (0x3000000)queueName        = 'MQOUT'
        (0x3000000)queueManagerName = 'MQSI_SAMPLE_QM'
        (0x3000000)replyIdentfier   = X'414d51204d5153495f53414d504c455f1f442f3b12600100'
        (0x3000000)msgId            = X'414d51204d5153495f53414d504c455f1f442f3b12600100'
        (0x3000000)correlId         = X'000000000000000000000000000000000000000000000000'
 (0x03000000):GroupId          = X'414d512042524f4b4552322020202020f59934620001803'
      )
    )
  )
)
```

ExceptionList
To trace data in the exception list, set Pattern to ${ExceptionList}.

You can also view message structure in the message flow, and other information, when you use the flow debugger.

Accessing databases from message flows

Create and configure message flows to access user databases.

Before you start:

Complete the following tasks:
- "Creating a message flow" on page 259
- Configuring broker and user databases

Read the following concept topic:
- "Message flow nodes" on page 6

Check which databases are supported on which platform, and if any restrictions apply:
- Supported databases

You can access information in a database to enhance or influence the operation of the message flow. You can also modify the contents of a database by inserting new information, or by removing or replacing existing information.

You can access a database from a message flow by using the following nodes:
- Compute
- Database
- DatabaseRetrieve
- DatabaseRoute
- DataDelete
- DataInsert
- DataUpdate
- Filter
- JavaCompute
- Mapping
- Warehouse

For more details about these nodes, and how to configure them in message flows, see "Built-in nodes" on page 875.

If you want the actions that the message flow takes against the database to be coordinated with other actions, configure the message flow to support global coordination of transactions. For information about how to complete this task, see "Configuring transactionality for message flows" on page 213.

To access a database from a message flow:

1. Identify the database that you want to access. You can access an existing database, or create a new database for this purpose. See "Data sources on z/OS" on page 1583 for more information about what to call a z/OS user database.

   If you want to create a new DB2® database, follow the instructions given in Creating the broker and user databases. If you want to use a database other than DB2, talk to your database administrator, or refer to the database product documentation for detailed instructions on how to complete this task.
2. Define a connection to the data source name (DSN) to enable a connection to the database, if one does not exist:
   a. Define a JDBC connection if you want to interact with a database directly from a Java application. You can code Java in both a JavaCompute node and in a Java user-defined node.

      For more information, see Enabling JDBC connections to the databases.
   b. Define an ODBC connection if you want to interact with a database in a node that supports ESQL, including a JavaCompute node in which you use the MbSQLStatement interface.

For more information, see Enabling ODBC connections to the databases.

3. Authorize the broker to access the database.

   Access to a user database from within a message flow is controlled by user ID and password.

   z/OS On z/OS, you can specify these values:

   - When you create the broker.

     The broker started task ID is used to access user databases, irrespective of the user ID and password specified on the mqsicreatebroker command in the BIPCRBK JCL in the customization data set `<hlq>.SBIPPROC`.

   - Optional: After you have created the broker.

     Use the BIPSDBP JCL in the customization data set `<hlq>.SBIPPROC` to customize the mqsisetdbparms command to specify a user ID and password for a specific database. This command changes the default values that you set when you created the broker (described earlier).

     You can create a user ID and password for any database (identified by a DSN) that is accessed by a message flow. You can, therefore, control access to a database at an individual level if you choose. This access includes databases that you have created and configured on distributed systems that are accessed by z/OS DB2 remote database access.

   Linux UNIX Windows On distributed systems, you can specify these values:

   - When you create the broker.

     The mqsicreatebroker command has two parameters, **-u DataSourceUserid** and **-p DataSourcePassword**, that you can use to identify the user ID that the broker uses to access its own database. If you specify these parameters, they are used as the default access control parameters for user databases that are accessed by message flows.

     If you do not specify **DataSourceUserid** and **DataSourcePassword**, the broker uses the values specified for the parameters **-i ServiceUserID** and **-a ServicePassword** (which identify the user under which the broker runs) as the default values.

   - Optional: After you have created the broker.

     Use the mqsisetdbparms command to specify a user ID and password pair. This command changes the defaults that you set when you created the broker (described earlier).

     You can create a user ID and password pair for any database (identified by a DSN) that is accessed by a message flow. You can therefore control access to a database at an individual level if you choose. This access includes databases that you have created and configured on z/OS that are accessed by brokers on distributed systems.

   If the user that created a table in a database is not the user that the broker is using to access the database, you must specify the user ID that created the database as the schema name in relevant ESQL statements, unless you have set up an alias or synonym.

If you access a database from a message flow using a Compute, Database, or Filter node, use the New Database Definition File wizard to enable a connection to the appropriate database. See “Adding database definitions to the workbench” on page 578 for further details.

The following samples access databases from message flows:

- Message Routing
- Data Warehouse
- Error Handler
- Airline Reservations

Message Routing and Data Warehouse use Compute nodes to access the database. Error Handler uses Database nodes to access the database, and Airline Reservations uses both Compute and Database nodes.

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

Accessing databases from ESQL

Configure your broker and your database to support connections from message flows.

Before you start:

- Create the broker.

You must configure both your broker and your databases to support read, write, and update operations in your message flows.

- Set the Data Source property of each node to the name (that is, the ODBC DSN) of the database that you want to access.
- Configure the broker to be able to connect to the database:
  - Create ODBC data source connections on the system on which the broker is running.
  - Define a user ID and password to be used by the broker to connect to the database:
    - If you have used the mqsisetdbparms command, or submitted the JCL member `BIPSDBP` in the customization data set `<hlq>.SBIPPROC` on z/OS, to set a user ID and password for a particular database, the broker uses these values to connect to the database.
    - If you have not set a specific user ID and password, the broker uses the default database user ID and password that you supplied on the mqsicreatebroker command, or modified by a subsequent mqsichangebroker command (or the equivalent console command or JCL on z/OS).
    - If you have not set a default database user ID and password, the broker uses its service ID and password.
- Set up the authorization for the user ID to access the database by using the administration facilities provided by the database vendor. If you do not do so, the broker generates an error when the message flow runs.
- All databases accessed from the same node must have the same ODBC functions as the database specified on the Data Source property on that node. This requirement is always satisfied if the databases are of the same type (for example, DB2 or Oracle), at the same release level (for example, release 9.1), and on the same platform. Other database combinations might have the same ODBC functions. If a node tries to access a database that does not have the same ODBC functions as the database specified on the Data Source property on that node, the broker generates an error message.
- With a single `SELECT FROM` clause, you can access only tables that exist in a single database.

- If you access database columns that have names composed of only numeric characters, you must enclose the names in double quotation marks; for example, "0001". Because of this restriction, you cannot use a `SELECT *` statement, which returns the names without quotation marks; the names are therefore invalid and the broker raises an exception.

For details of the ESQL statements and functions that you can use to access databases, see "Interaction with databases using ESQL" on page 366.

Configuring transactionality for message flows

A message flow runs in a single transaction, which is started when data is received by an input node, and can be committed or rolled back when all processing has completed.

Before you start:

Read "Message flow transactions" on page 136 to ensure that you understand how the broker handles transactions.

You must have created a message flow before you complete this task.

How individual nodes, and the message flow itself, participate in transactions depends on the way you design and develop the message flow, and the level of additional configuration you perform:

1. Configure the node properties in your message flow to set the required level of participation in transactions.
2. If you want the updates made by the message flow to be globally coordinated by an external transaction manager, configure the message flow properties.

When you have finished message flow design and development, you can deploy the BAR file to the broker or brokers on which you want the message flow to run. However, if you have configured your message flows for globally coordinated transactions, additional configuration might be required. You, or your system administrator, must ensure that your broker environment, the transaction manager, and the participating resource managers are all correctly configured to support coordinated transactions, before you run the message flow. For details of what might be required, see Configuring global coordination of transactions (two-phase commit).

If the broker environment, the transaction manager, and the external resource managers are not correctly configured for global coordination, the message flow transactions will not be globally coordinated.

Configuring node properties

You can configure the nodes in your message flow to determine how the work taken by each node participates in the message flow transaction. Most nodes for which transactionality is relevant have one or more properties that you can configure to dictate behavior. Therefore, you can decide for each individual node whether it participates in the message flow transaction, or operates independently. Typically, these properties include an option of Automatic, so that subsequent nodes in the flow assume the characteristics set by the input node.

Nodes that support transports that cannot participate in transactions might have other properties to determine what the broker does when a message flow failure occurs. For example, the FileInput node has a set of Retry properties that you can set to determine failure behavior.

A few nodes that interact with external resources do not provide properties; typically, these nodes are included in the message flow transactions, but some exceptions exist; you must check the section that describes properties and how to set them, for each node that you include in your flow to ensure that you understand what action is taken.

If you configure a node not to participate in the message flow transaction, the actions that it takes are committed, or rolled back, when the node exits. No further action is taken when the flow itself completes.

To configure message flow behavior by setting node properties:

1. Switch to the Broker Application Development perspective.
2. Open the message flow that you want to configure.
3. Set the Transaction mode property for the input nodes in this message flow. The value that you set determines the behavior of the input node, and sets the default behavior for the rest of the message flow. Typically, you can choose the value Yes or No;
   - Yes means that the input node completes its own operation under sync point, and the default behavior in the message flow is for actions to be taken under sync point.
   - No means that the input node completes its own operation out of sync point, and the default behavior in the message flow is for actions to be taken out of sync point.

   Some nodes have additional or alternative values; for example, you can set the property on the MQInput node to Automatic, which means that the node gets the message under sync point if the message is persistent, and out of sync point if it is non-persistent.

   For details of the specific options for and actions taken by each node, see the relevant node description; the properties, the tabs they are defined on, and the resulting behavior are not identical across all input nodes.
4. If your message flow includes nodes that interact with external resources, including output, request, and reply nodes, you can set a transaction property on most of these nodes.

   Set the property only if you want to change the behavior of the individual node from the default behavior for the message flow, which you set on the input node. The value that you set on this node has no effect on subsequent nodes in the message flow. If the node does not have a transactional property, its behavior is governed by the default behavior for the message flow, which you set in the input node.

   If your message flow is updating a database from multiple nodes in a single message flow, read the conceptual information about message flow transactions to understand the possible interactions.

   a. Set the Transaction property for each node, if supported.
   b. Set the properties that define how errors are handled, if supported. For example, for nodes like the Compute node that can access databases, set the Treat warnings as errors and Throw exception on database error properties to define how that node handles database warnings and errors. Whether

you select these properties, and how you connect the failure terminals of the nodes, also affect the way in which database updates are committed or rolled back.

Configuring message flow properties

When you have configured your message flow, you must add it to a BAR file before you can deploy it. When you add it to a BAR file, the message flow is compiled, and additional properties are available for configuration.

The most important property concerned with transactions on distributed systems is Coordinated Transaction. By default, this property is cleared (not selected), which means the message flow is partially coordinated and the broker commits or rolls back the message flow transaction. If you select this property, the input node calls the external transaction manager WebSphere MQ for commit and rollback processing.

This property is ignored when the message flow is deployed to a broker that is running on a z/OS system.

To configure message flow properties:

1. Add the message flow to a broker archive.
2. Select the **Manage and Configure** tab below the broker archive editor view, and select the message flow. The configurable properties for the message flow in the broker archive are displayed in the **Properties** view.

   Select coordinatedTransaction to configure the message flow as globally coordinated; when you set this property, the external transaction manager (WebSphere MQ) coordinates the transaction with all the resource managers that you have defined to the queue manager.

   z/OS On z/OS, transactions are always globally coordinated. The setting of the coordinatedTransaction property for a message flow is ignored. Coordination is provided by the transaction manager RRS.

Configuring JMSInput and JMSOutput nodes to support global transactions

If you want to include JMSInput and JMSOutput nodes in globally coordinated transactions, you must complete additional configuration.

If you require transaction coordination, choose a JMS provider that conforms to the Java Message Service Specification, version 1.1 and that supports the JMS XAResource API through the JMS session.

If the message designer has specified a non-XA-compliant provider, the non-transactional mode only is supported. In this case, you must set the Transaction mode property to None for all JMSInput and JMSOutput nodes.

To configure JMSInput and JMSOutput nodes:

1. Switch to the Broker Application Development perspective.
2. Set the message flow property Coordinated Transaction to yes in the BAR file properties.
3. For each JMSInput or JMSOutput node required in the global transaction, set the Advanced property Transaction mode to Global in the message flow editor.

4. Create a Queue Connection Factory and either use the default name, *recoverXAQCF* , or supply your own name. See the JMSInput or JMSOutput node for further details about creating JNDI administered objects.
5. On distributed systems, you must set up a stanza for each JMSProvider that you want to use, before deployment.

   The following table shows the JMSProvider switch files that are provided on each platform.

Platform	32-bit file	64-bit file
AIX®	`libJMSSwitch.so`	`libJMSSwitch64.so`
HP-Itanium®		`libJMSSwitch.so`
HP-UX on PA-RISC	`libJMSSwitch.sl`	`libJMSSwitch64.sl`
Linux on POWER®		`libJMSSwitch.so`
Linux on System z®		`libJMSSwitch.so`
Linux on x86	`libJMSSwitch.so`	
Linux on x86-64	`libJMSSwitch.so`	`libJMSSwitch64.so`
Solaris on SPARC	`libJMSSwitch.so`	`libJMSSwitch64.so`
Solaris on x86-64		`libJMSSwitch.so`
Windows	`JMSSwitch.dll`	

   Select the appropriate link for details of this task on the platform, or platforms, that your enterprise uses:

   - Linux UNIX Linux and UNIX systems
   - Windows Windows systems

     On Windows only, you must also modify the queue manager authorization.

   For further information, see:

   - "Configuring for coordinated transactions" on page 1022 within the JMSInput node topic
   - "Configuring for coordinated transactions" on page 1037 within the JMSOutput node topic

   z/OS On z/OS, the only JMSProvider supported is the IBM WebSphere MQ Java Client, and the only transport mode supported for that client is BIND mode; no further configuration steps are required.

The JMS provider might supply additional JAR files that are required for transactional support; see the documentation supplied with the JMS provider for more information. For example, on distributed systems, the WebSphere MQ JMS provider supplies an extra JAR file `com.ibm.mqetclient.jar`.

You must add any additional JAR files to the broker `shared_classes` directory:

- Linux UNIX On Linux and UNIX: `var/mqsi/shared-classes`.
- Windows On Windows, `%ALLUSERSPROFILE%\Application Data\IBM\MQSI\shared-classes` where `%ALLUSERSPROFILE%` is the environment variable that defines the system working directory. The default directory depends on the operating system:
  - On Windows XP and Windows Server 2003: `C:\Documents and Settings\All Users\Application Data\IBM\MQSI\shared-classes`

– On Windows Vista and Windows Server 2008: `C:\ProgramData\IBM\MQSI\shared-classes`

The actual value might be different on your computer.

For more information, see the section on making the JMS provider client available to the JMS nodes in "JMSInput node" on page 1020.

Optional: If you want to secure the JMS connection factory, the JNDI bindings, or both, see "Securing JMS connections and JNDI lookups" on page 221.

Linux and UNIX systems: configuring the queue manager to coordinate JMS resources

Define a stanza in the broker's queue manager `qm.ini` file for each new JMS provider, where the JMS provider can be specified by an JMSInput or JMSOutput node included in a message flow that is running on the broker.

The parameters that are defined in `XAOpenString` are comma delimited and positional. Represent missing optional parameters by a comma if you include other parameters later in the string.

The following stanza entry is an example that you can add when using WebSphere MQ Java as the JMS provider:

```
XAResourceManager:
    Name=WBIWMQJMS
    SwitchFile=/install_dir/lib/JMSSwitch.so
    XAOpenString=<Initial Context Factory>,
              <location of JNDI bindings>'
              <LDAP Principal>,
              <LDAP Credentials>,
              <Recovery Connection Factory Name>,
              <JMS Principal>,
              <JMS Credentials>
              ThreadOfControl=THREAD
```

where:

install_dir
: Is the location of the WebSphere Message Broker installation. This value is mandatory where the LDAP parameters are omitted, but a user-defined Queue Connection Factory is specified for recovery.

<Initial Context Factory>
: Is the Initial Context Factory identifier for the JMS provider; this value is required.

<Location of JNDI bindings>
: Is either the file path to the bindings file, or the LDAP directory location of the JNDI administered objects that can be used to create an initial context factory for the JMS connection. When supplying the file path to the bindings file, do not include the file name. See the JMSInput or JMSOutput node for further details on creating the JNDI administered objects; this value is required.

<LDAP Principal>
: Is an optional parameter used to specify the principal (user ID) that might be required when an LDAP database is used to hold the JNDI administered objects.

<LDAP Credentials>
Is an optional parameter used to specify the Credentials (password) that might be required if a password protected LDAP database is used to hold the JNDI administered objects.

<Recovery Connection Factory Name>
Is an optional parameter used to specify the name of a Queue Connection Factory object in the JNDI administered objects for recovery purposes, when the non default name is required.

<JMS Principal>
Is an optional parameter for the user ID required to connect to a JMS provider, using a secure JMS Connection Factory.

<JMS Credentials>
Is an optional parameter for the password required to connect to the same JMS provider in conjunction with the JMS principal.

Switch files are installed in the *install_dir* /lib directory. To simplify the contents of the qm.ini file, create a symbolic link to the switch file for the queue manager to retrieve from /var/mqm/exits (for 32-bit brokers) or /var/mqm/exits64 (for 64-bit brokers). For example:

```
ln -s install_dir/lib/libJMSSwitch.so /var/mqm/exits/JMSSwitch

ln -s install_dir/lib/libJMSSwitch.so /var/mqm/exits64/JMSSwitch
```

If you create a link for both 32-bit and 64-bit switch files on a single computer, ensure that you specify the same name in /exits and in /exits64, as shown in the example.

The values for the Initial Context factory and Location of JNDI bindings in the stanza must match the values that you specified in the JMSInput or JMSOutput nodes in the message flows.

All LDAP parameters must match the values that you specified on the mqsicreatebroker or mqsichangebroker command.

The Recovery Factory Name must match a Queue Connection Factory name that is created in the JNDI administered objects. If you do not specify a name, a default factory called recoverXAQCF is used. In either case, this value must refer to a JNDI administered object that has already been created.

The JMS Principal and JMS Credentials must be configured together.

The following example shows the format of a stanza in the qm.ini file that describes a JMS provider for global transactions:

```
XAResourceManager:
    Name=XAJMS_PROVIDER1
    SwitchFile=/opt/var/mqsi/lib/JMSSwitch.so
    XAOpenString= com.sun.jndi.fscontext.RefFSContextFactory,
              /Bindings/JMSProvider1_Bindings_Directory,
              ,
              ,
              ,
              myJMSuser1,
              passwd
              ThreadOfControl=THREAD
```

where:

XAJMS_PROVIDER1
Is the user-defined name for the resource manager

/opt/var/mqsi
Is the <Installation Path>

com.sun.jndi.fscontext.RefFSContextFactory
Is the <Initial Context Factory>

/Bindings/JMSProvider1_Bindings_Directory
Is the location of the bindings

myJMSuser1
Is the <JMS Principal>

passwd
Is the password used in <JMS Credentials>

In this example, the optional fields `<LDAP Principal>`, `<LDAP Credentials>`, and `<Recovery Connection Factory Name>` are not required, therefore the positional comma delimiters only are configured in the `XAOpenString` stanza.

Windows systems: configuring the queue manager to coordinate JMS resources

Use WebSphere MQ Explorer to configure the XA resource managers for the queue manager.

Complete the following steps:

1. Open WebSphere MQ Explorer.
2. Select the queue manager for your broker and click **Properties**.
3. Select **XA resource managers** in the left pane and click **Add**.
4. Complete the fields to define a new resource manager:
   - **Name**: Enter the name of the resource manager; for example, WBIWMQJMS.
   - **SwitchFile**: Enter the full path of the switch file; for example, *install_dir*`\bin\JMSSwitch.dll`.
   - **XAOpenString**: Enter the following values, which are comma delimited and positional. Represent missing optional parameters by a comma if you include other parameters later in the string.

     Initial Context Factory
     The Initial Context Factory identifier for the JMS provider; this value is required.

     Location of JNDI bindings
     Either the file path to the bindings file, or the LDAP directory location of the JNDI administered objects that can be used to create an initial context factory for the JMS connection. If you supply the file path to the bindings file, do not include the file name. See the JMSInput or JMSOutput node for further details about creating the JNDI administered objects; this value is required.

     LDAP Principal
     Optional: The principal (user ID) that might be required when an LDAP database is used to hold the JNDI administered objects.

     LDAP Credentials
     Optional: The credentials (password) that might be required if a password protected LDAP database is used to hold the JNDI administered objects.

Recovery Connection Factory Name
Optional: The name of a Queue Connection Factory object in the JNDI administered objects for recovery purposes, when the non default name is required.

JMS Principal
The user ID that is required to connect to a JMS provider, using a secure JMS Connection Factory.

JMS Credentials
The password that is required to connect to the same JMS provider in conjunction with the JMS principal.

The values for the Initial Context factory and Location of JNDI bindings in the stanza must match the values that you specified in the JMSInput or JMSOutput nodes in the message flows.

All LDAP parameters must match the values that you specified on the mqsicreatebroker or mqsichangebroker command.

The Recovery Factory Name must match a Queue Connection Factory name that is created in the JNDI administered objects. If you do not specify a name, a default factory called `recoverXAQCF` is used. In either case, this value must refer to a JNDI administered object that has already been created.

The JMS Principal and JMS Credentials must be configured together.

- **XACloseString**: Leave this field blank.
- **ThreadOfControl**: Set the value Thread.

5. Click **OK** to complete the XA resource manager definition.
6. Click **OK** to close the queue manager properties dialog.
7. Click **File** → **Exit** to close WebSphere MQ Explorer.
8. Copy the switch file (for example, `JMSSwitch.dll`) to the `\exits` subdirectory in the WebSphere MQ installation directory.

Next: modify the queue manager authorization.

Windows systems: modifying the queue manager authorization

Authorize the broker and queue manager to access shared resources that are associated with the JMSProvider.

Before you start, set up your JMSProvider configurable service; see "Making the JMS provider client available to the JMS nodes" on page 1021 (within the JMSInput node topic) or "Making the JMS provider client available to the JMS nodes" on page 1034 (within the JMSOutput node topic).

Complete the following steps on the Windows system on which the broker is running:

1. If you defined the broker queue manager when you created the broker by running the mqsicreatebroker command, the two components share the same administrative ID, defined as the broker service ID, and you do not have to take any further action.
2. If you specified an existing queue manager when you created the broker, check that its administrative ID is the same ID as that used for the service ID of the broker. If the ID is not the same, change the queue manager ID to be the same as the broker service ID:
   a. Click **Start** → **Run** and enter dcomcnfg. The Component Services window opens.

b. In the left pane, expand **Component Services** → **Computers** → **My Computer** and click **DCOM Config**.
c. In the right pane, right-click the WebSphere MQ service labelled **IBM MQSeries Services**, and click **Properties**.
d. Click the **Identity** tab.
e. Select **This user** and enter the user ID and password for the broker service ID to associate that ID with the queue manager.
f. Click **OK** to confirm the change.

Securing JMS connections and JNDI lookups

If you want additional security for JMS connectivity and your JMSInput, JMSOutput, and JMSReply nodes, two configuration options are supported.

When you include JMS nodes in your message flow, you can optionally secure JMS connection resources. You can secure one, both, or neither of these options, depending on the level of security and access that you want to enforce.

1. To secure a JMS connection:
   a. Specify the Connection Factory Name property on the JMS node. You must set this property for every JMS node.
   b. Use the mqsisetdbparms command to authorize the user ID and password for the specified connection factory. For example:

      ```
      mqsisetdbparms MyBroker1 -n jms::tcf1 -u myuserid -p secret
      ```

      where `tcf1` is the name of the connection factory that matches the node property that you set.
2. To secure JNDI bindings lookups:
   a. Specify the Initial Context Factory property on the JMS node. You must set this property for every JMS node.
   b. Use the mqsisetdbparms command to authorize the user ID and password for the specified context factory. For example:

      ```
      mqsisetdbparms MyBroker1 -n jndi::com.sun.jndi.fscontext.RefFSContextFactory
           -u myuserid -p secret
      ```

      where `com.sun.jndi.fscontext.RefFSContextFactory` is the name of the initial context factory that you set.

Configuring the broker to enable a JMS provider's proprietary API

Some JMS providers provide an alternative interface to the standard JMS specification for particular JMS API calls. In these cases, IBM supplies a Java class to interface with that proprietary API.

For example, BEA WebLogic uses a component called a *Client Interposed Transaction Manager* to allow a JMS client to obtain a reference to the XAResource that is associated with a user transaction.

If the WebSphere Message Broker JMS nodes use BEA WebLogic as the JMS provider, and the nodes must participate in a globally coordinated message flow, you must modify the configurable services properties that are associated with that vendor. The following table shows the properties that have been added to the configurable service for BEA WebLogic.

JMS provider	Property	Purpose	Default value
BEA_WebLogic	proprietaryAPIHandler	The name of the IBM supplied Java class to interface with a JMS provider's proprietary API.	`com.ibm.broker.apihandler. BEAWebLogicAPIHandler`
	proprietaryAPIAttr1	The Initial Context Factory class name for the vendor	`weblogic.jndi. WLInitialContextFactory`
	proprietaryAPIAttr2	The URL of the WebLogic bindings	*URL JNDI bindings*
	proprietaryAPIAttr3	The DNS name of the JMS server	*Server name*

In the list of JMS provider configurable services, the name of the IBM supplied Java class is set to the default value for the proprietaryAPIHandler property. Typically, you do not need to change this value, unless you are instructed to do so by an IBM Service team representative.

- Use the mqsichangeproperties command to modify values of the properties for this JMS provider.

  The following example shows how to change the values of the properties proprietaryAPIAttr2 and proprietaryAPIAttr3 for the JMS provider configurable service definition called BEA_Weblogic, where these properties represent the URL of the WebLogic bindings and the DNS Server name of the BEA WebLogic JMS Server:

  ```
  mqsichangeproperties WBRK61_DEFAULT_BROKER -c JMSProviders -o BEA_Weblogic
  -n proprietaryAPIAttr2,proprietaryAPIAttr3 -v t3://9.20.94.16:7001,BEAServerName
  ```

- Use the mqsireportproperties command to display the properties for a JMS provider.

  The following example shows how to display the properties for all the broker's JMS provider resources (the default JMS provider resources and those configurable services that are defined with the mqsicreateconfigurableservice command):

  ```
  mqsireportproperties WBRK61_DEFAULT_BROKER -c JMSProviders -o BEA_WebLogic –r
  ```

  The result of this command has the following format:

  ```
  ReportableEntityName=''
  JMSProviders
    BEA_Weblogic=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
      proprietaryAPIAttr1='weblogic.jndi.WLInitialContextFactory'
      proprietaryAPIAttr2='t3://9.20.94.16:7001'
      proprietaryAPIAttr3='BEAServerName'
      proprietaryAPIAttr4='default_none'
      proprietaryAPIAttr5='default_none'
      proprietaryAPIHandler='com.ibm.broker.apihandler.BEAWebLogicAPIHandler'
  ```

  The default location for the JMS provider JAR files is the broker's shared-classes directory. You can specify an alternative location for the JAR files by using the mqsichangeproperties command, as shown in the following example:

  ```
  mqsichangeproperties WBRK61_DEFAULT_BROKER -c JMSProviders -o BEA_WebLogic -n jarsURL
  -v /var/mqsi/WebLogic
  ```

  On Windows, the file location cannot be a mapped network drive on a remote Windows computer; the directory must be local or on a Storage Area Network (SAN) disk.

- Use the mqsicreateconfigurableservice command to add a JMS provider.

The following example shows how to add a JMS provider called BEAV91 for broker WBRK61_DEFAULT_BROKER, specifying the name of an IBM supplied Java class called `com.ibm.broker.apihandler.BEAWebLogicAPIHandler` to handle vendor-specific API calls:

```
mqsicreateconfigurableservice WBRK61_DEFAULT_BROKER -c JMSProviders -o BEAV91
-n proprietaryAPIHandler,proprietaryAPIAttr1,proprietaryAPIAttr2,proprietaryAPIAttr3
–v com.ibm.broker.apihandler.BEAWebLogicAPIHandler,weblogic.jndi.WLInitialContextFactory,
t3://9.20.94.16:7001,BEAServerName
```

- If you have defined a user-defined JMS provider configurable service, set the value for the proprietaryAPIHandler property manually.

Changing connection information for the IMSRequest node

You can create a configurable service that the IMSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name, performance, and security values without needing to redeploy your message flow.

Before you start:

- Read "Configurable services" on page 66 to find out more about configurable services.
- Read "IBM Information Management System (IMS)" on page 59 for background information.

Use the IMSConnect configurable service to change the connection information for the IMSRequest node. Two configurable services can connect to the same instance of IMS Connect. The properties of the IMSConnect configurable service are described in Configurable services properties.

Creating, changing, reporting, and deleting configurable services

- To create a configurable service, use the mqsicreateconfigurableservice command, as shown in the following example. This example creates an IMSConnect configurable service for the IMS instance IMSA that is running on `test.ims.ibm.com` port 9999:

  ```
  mqsicreateconfigurableservice WBRK61_DEFAULT_BROKER -c IMSConnect -o myIMSConnectService
  -n Hostname,PortNumber,DataStoreName -v test.ims.ibm.com,9999,IMSA
  ```

- To change a configurable service, use the mqsichangeproperties command, as shown in the following example. This example changes all the nodes that are configured to use the *myIMSConnectService* configurable service. After you run this command, the IMSRequest node connects to the production system (`production.ims.ibm.com`) instead of the test system (`test.ims.ibm.com`).

  ```
  mqsichangeproperties WBRK61_BROKER -c IMSConnect -o myIMSConnectService -n Hostname
  -v production.ims.ibm.com
  ```

- To display all IMSConnect configurable services, use the mqsireportproperties command, as shown in the following example:

  ```
  mqsireportproperties WBRK61_DEFAULT_BROKER -c IMSConnect -o AllReportableEntityNames -r
  ```

- You can delete a configurable service that you have created by using the mqsideleteconfigurableservice command, as shown in the following example:

  ```
  mqsideleteconfigurableservice WBRK61_DEFAULT_BROKER -c IMSConnect -o myIMSconnectService
  ```

Configuring message flows for data conversion

If you exchange messages between applications that run on systems that are incompatible in some way, you can configure your system to provide data conversion as the message passes through the broker.

Data conversion might be necessary if either of the following two values are different on the sending and receiving systems:

1. CCSID. The Coded Character Set Identifier refers to a set of coded characters and their code point assignments. WebSphere Message Broker can process and construct application messages in any code page for which WebSphere MQ provides conversion to and from Unicode, on all operating systems. For more information about code page support, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.

   This behavior might be affected by the use of other products in conjunction with WebSphere Message Broker. Check the documentation for other products, including any databases that you use, for further code page support information.
2. Encoding. This setting defines the way in which a machine encodes numbers; that is, binary integers, packed-decimal integers, and floating point numbers. Numbers that are represented as characters are handled in the same way as all other string data.

If the native CCSID and encoding on the sending and receiving systems are the same, you do not need to call data conversion processes.

WebSphere Message Broker and WebSphere MQ provide data conversion facilities to support message exchange between unlike systems. Your choice of which facilities to use depends on the characteristics of the messages that are processed by your message flows:

- Messages that contain text only
- Message that include numerics
- Messages that are self-defining

Messages that contain text only

Read this section if your messages are WebSphere MQ messages that contain all text (character data or string).

If WebSphere MQ supports the systems on which both sending and receiving applications are running for data conversion, use WebSphere MQ facilities which provide the most efficient data conversion option.

The default behavior of WebSphere MQ is to put messages to queues specifying the local system CCSID and encoding. Applications issuing MQGET can request that the queue manager provides conversion to their local CCSID and encoding as part of get processing.

To use this option:

1. Design messages to be text-only. If you are using COBOL, move numeric fields to USAGE DISPLAY to put them into string form.
2. Set the Format field in the MQMD to MQFMT_STRING (value MQSTR).

3. Call MQGET with MQGMO_CONVERT in the receiving application. If you prefer, you can convert when the message is received by the broker, by setting the Convert property of the MQInput node to yes (by selecting the check box).

If you require more sophisticated data conversion than WebSphere MQ provides in this way (for example, to an unsupported code page), use WebSphere MQ data conversion exits. For more information about these, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.

Messages that include numerics

Read this section if your messages include numeric data, or are text only but are not WebSphere MQ messages.

If these messages can be predefined (that is, their content and structure is known and predictable), use the facilities provided by WebSphere Message Broker and the MRM.

All application messages are handled by the broker in Unicode, to which they are converted on input, and from which they are converted on output. You can configure message flows to influence the way in which output messages are constructed.

To use this option:

1. Define the output message in the MRM domain. You can create this definition in one of the following ways:
   - Import an external message definition (for example a C header or COBOL copybook).
   - Create the message model in the message definition editor.
2. Configure a message flow to receive and process this message:
   a. If you include an MQInput node, do not request conversion by this node.
   b. Include a Compute node in the message flow to create the output message with the required content:
      - If the output message is a WebSphere MQ message, code ESQL in the Compute node to set the CCSID and encoding for the target system in the MQMD.

        For example, to set values for a target z/OS system running with CCSID of 37 and encoding of 785:

        ```
        SET OutputRoot.MQMD.CodedCharSetId = 37;
        SET OutputRoot.MQMD.Encoding = 785;
        ```

      - If the output message is not a WebSphere MQ message, code ESQL in the Compute node to set the CCSID and encoding for the target system in the Properties folder.

Messages that are self-defining

Read this section if your messages are self-defining.

Self-defining messages are supported in the XML and JMS domains. These messages are all text and can be handled by WebSphere MQ, if they originate from, or are destined for, WebSphere MQ applications. If not, use WebSphere Message Broker facilities by setting the CCSID and Encoding fields in the Properties folder in the message when it passes through a Compute node.

Using MQGet nodes

The MQGet node processes messages in a particular way, and you can use it in request-response message flows.

How the MQGet node processes messages

The MQGet node processes each message that it receives.

This topic contains the following sections:

Propagating the message

1. If an MQ Message Descriptor header (MQMD) is present in the input tree, the MQGet node uses it. If not, the node creates a default MQMD.
2. The node also creates a default MQ Get Message Options (MQGMO) structure based on the values that you have set for the node properties. If an MQGMO is present in the input tree, the node uses its content to modify the default one.

   When you include an MQGMO to override the default one, you must specify all the options that you are replacing. For example, if you set the option field to MQGMO_CONVERT, that value overrides all options that you set with the workbench. If you do not use an overriding MQGMO, WebSphere Message Broker uses the following values:
   - If Wait interval is not zero, MQGMO_WAIT is set; otherwise, MQGMO_NOWAIT is used.
   - If Transaction mode is set to Yes, MQGMO_SYNCPOINT is used.
   - If Transaction mode is set to No, MQGMO_NOSYNCPOINT is used.
   - If Transaction mode is set to Automatic, MQGMO_SYNCPOINT_IF_PERSISTENT is used.
   - The only other option that is used by default in the node properties is MQGMO_COMPLETE_MSG, which is set if Transaction mode is set to Yes or No. This option is not set when your broker is running on z/OS.
   - No other options are used by default.
3. The node makes the MQGet call to WebSphere MQ.
4. The node analyzes the completion code (CC), and propagates the message to the appropriate terminal:

   OK The node creates the output LocalEnvironment and the output message trees using standard message-parsing techniques, then propagates the message to the Out terminal.

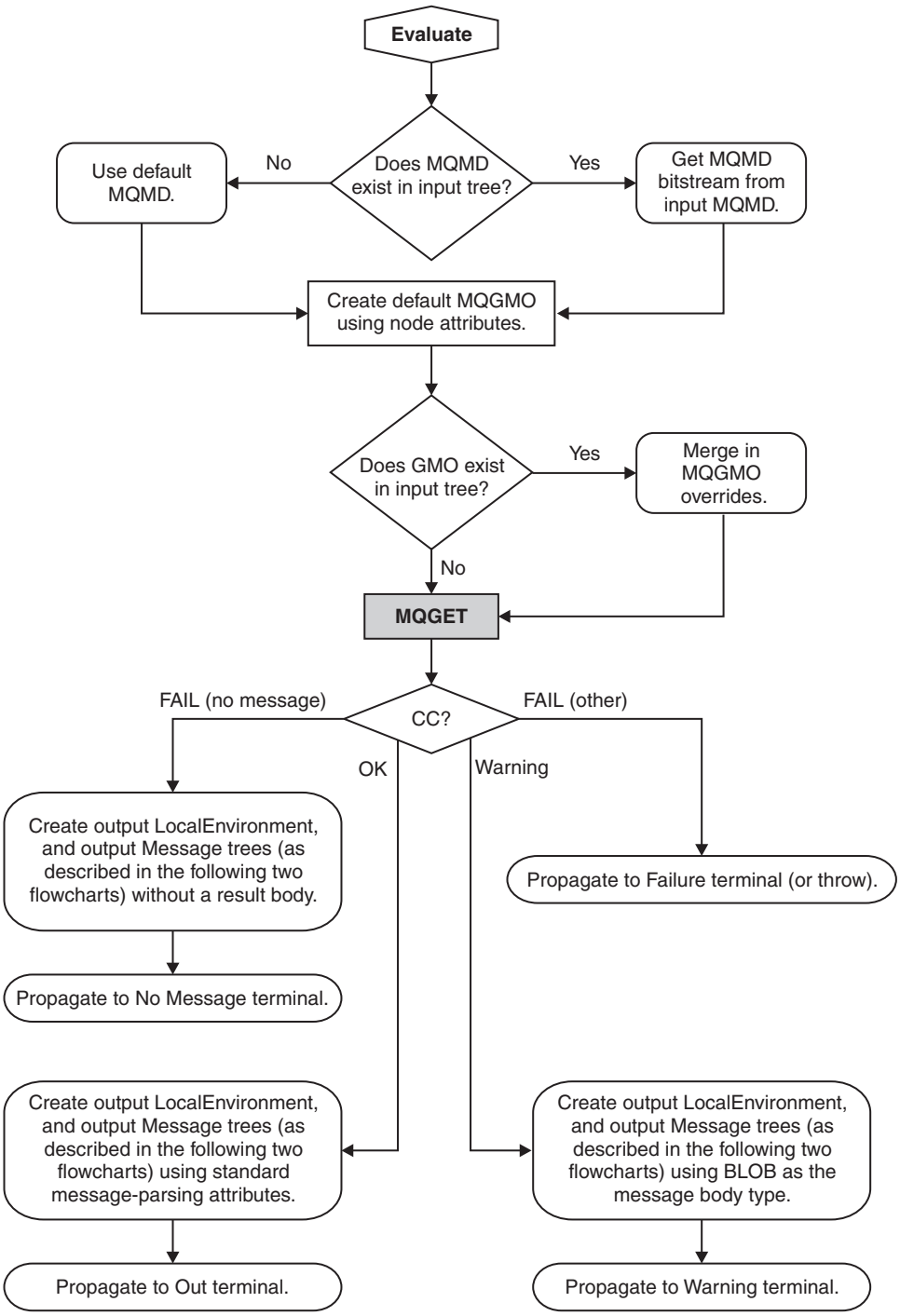
   Warning
   The node creates the output LocalEnvironment and the output message trees using BLOB as the message body type, then propagates the message to the Warning terminal, if it is connected. If the Warning terminal is not connected, no propagation occurs, and the flow ends.

   Fail (no message)
   The node creates the output LocalEnvironment and the output message trees by copying the input trees, then propagates the message to the No Message terminal, if it is connected. If the No Message terminal is not connected, no propagation occurs. The output message that is propagated to the No Message terminal is constructed from the input

message only, according to the values of the Generate Mode property, and the Copy Message or Copy Local Environment properties.

Fail (other)
The node propagates the message to the Failure terminal. If the Failure terminal is not connected, the broker throws an exception and returns control to the closest upstream node that can process it. For more information, see "Handling errors in message flows" on page 244.

The following diagram shows this processing:

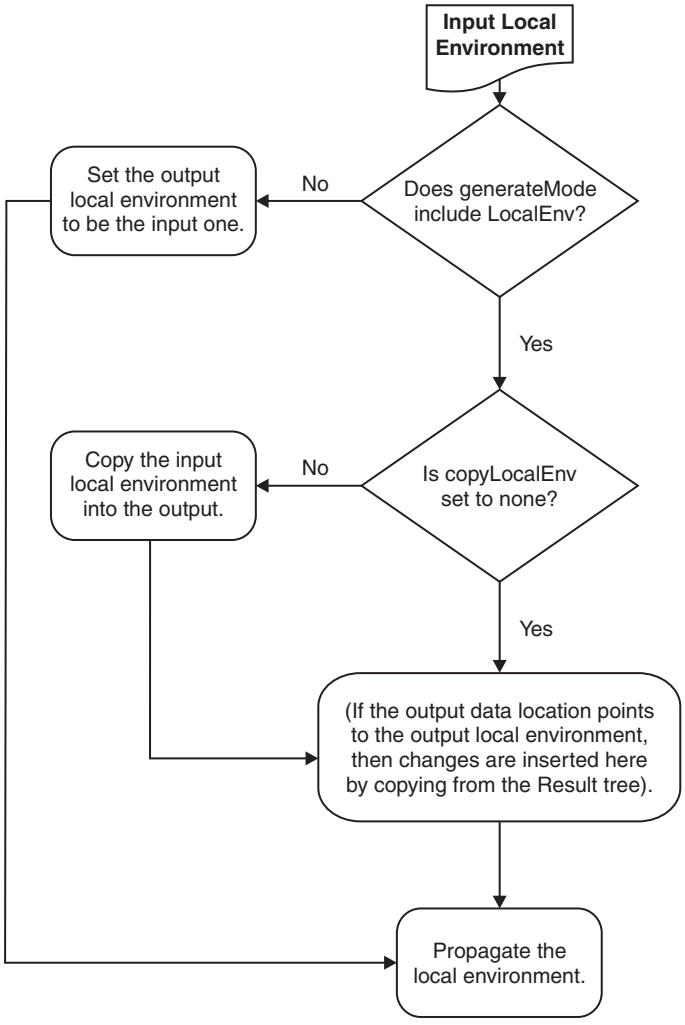


Constructing OutputLocalEnvironment

1. If the Generate Mode property on the MQGet node is set to an option that does not include LocalEnvironment, the node copies the input local environment tree to the output local environment tree.

   If this copy is made, any updates that are made in this node to the output local environment tree are not propagated downstream.
2. If the Copy Local Environment property is set to an option other than None, the node copies the input local environment tree to the output local environment tree.
3. If the output data location points to the output local environment tree, the node applies changes in that tree by copying from the result tree.
4. The local environment tree is propagated.

The following diagram shows this processing:

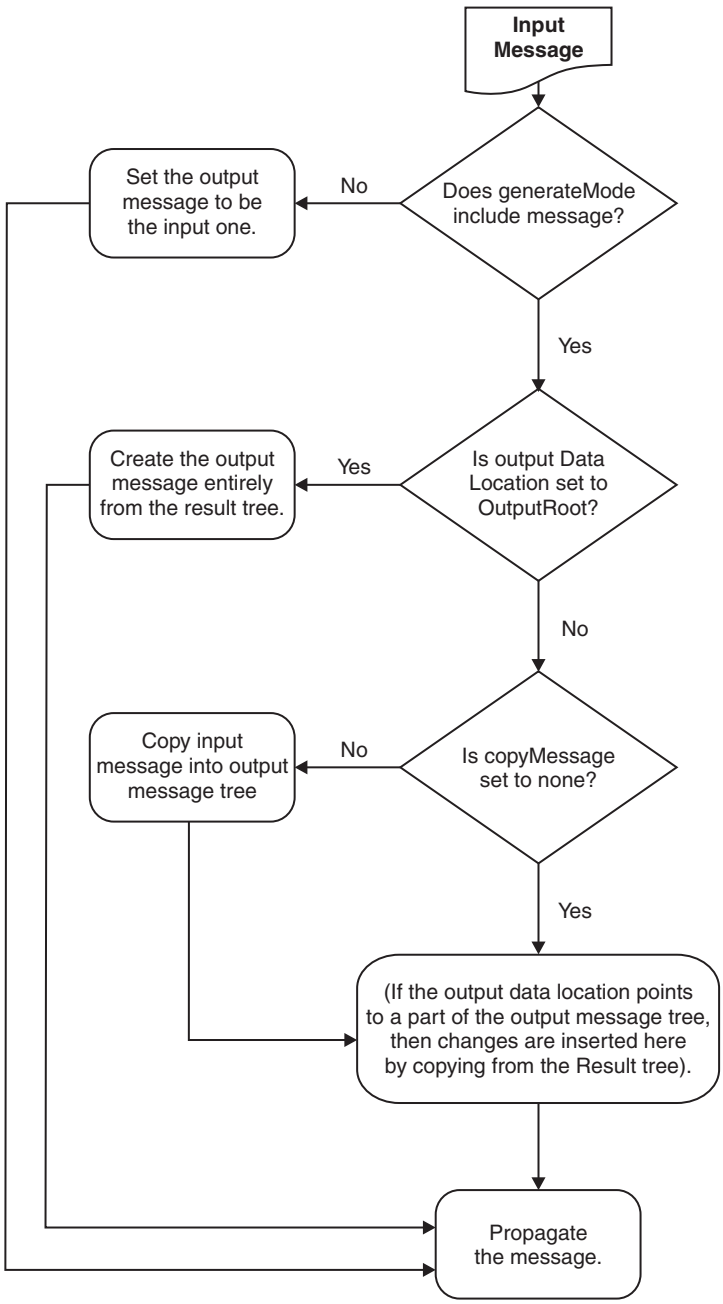


Constructing the Output message

1. If the Generate Mode property on the MQGet node is set to an option that does not include Message, the node copies the input message tree to the output message tree. Go to step 5.

2. If the Output Data Location property is set to OutputRoot, the node creates the output message tree entirely from the result tree. Go to step 5.
3. If the Copy Message property is set to a value other than None, the node copies the input message tree to the output message tree.
4. If the Output Data Location property points to a part of the output message tree, the node applies changes in that tree by copying from the result tree at the point that is defined by the Result Data Location property.
5. The message tree is propagated.

The following diagram shows this processing:



For an example of how this processing is implemented in a message flow, see "A request-response scenario that uses an MQGet node."

A request-response scenario that uses an MQGet node

Read about a scenario in which an MQGet node is used in a request-response flow, and how the node processes the input messages to construct the output messages, based on both the content of the local environment tree and the input parameters that you set.

A request-response flow is a specialized form of a point-to-point application. For a general description of these applications, see Application communication models. For an example of a request-response message flow, see the following sample:

- Coordinated Request Reply

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

You can include an MQGet node anywhere in a message flow, including a flow that implements a request-response scenario. The MQGet node receives an input message on its input terminal from the preceding node in the message flow, issues an MQGET call to retrieve a message from the WebSphere MQ queue that you have configured in its properties, and builds a result message tree. Finally, it uses the input tree and the result tree to create an output tree that is then propagated to its Output, Warning, or Failure terminal, depending on the configuration of the node and the result of the MQGET operation.

How the MQGet node handles the local environment:

The MQGet node examines the local environment tree that is propagated from the preceding node, uses the content that is related to the MQGMO (MQ Get Message Options) and the MQMD (MQ Message Descriptor header), and updates the local environment:

- The node reads the MQGMO structure from `${inputMQParmsLocation}.MQGMO.*`.
- The node copies the WebSphere MQ completion and reason codes to `${outputMQParmsLocation}.CC` and `${outputMQParmsLocation}.RC`.
- The node writes the complete MQGMO that is used for the MQGET call into `${outputMQParmsLocation}.MQGMO` if `${inputMQParmsLocation}.MQGMO` exists in the input tree.
- The node writes the MQMD that is passed to the MQGET call (that contains the values that are specified in the input message or are generated by the node) into `${inputMQParmsLocation}.MQMD`, deleting any existing content.

Set the value to `${inputMQParmsLocation}` in the MQGet node property Input MQ Parameters Location on the **Request Properties** tab.

Set the value to `${outputMQParmsLocation}` in the MQGet node property Output MQ Parameters Location on the **Result Properties** tab.

For more information about these properties, see "MQGet node" on page 1069.

In summary:

${inputMQParmsLocation}

- *QueueName*: Optional override for MQGet node *Queue Name* property

- *InitialBufferSize*: Optional override for MQGet node *Initial Buffer Size* property
- *MQGMO.**: Optional MQGET message options that are used by the MQGet node

${outputMQParmsLocation}

- *CC*: MQGET call completion code
- *RC*: MQGET call result code
- *MQGMO.**: MQGET message options that are used if present in `${inputMQParmsLocation}`
- *MQMD*: unparsed MQ Message Descriptor for received messages[1]
- *Browsed*: Set to true if the message is browsed. Not present if the message is removed from the queue

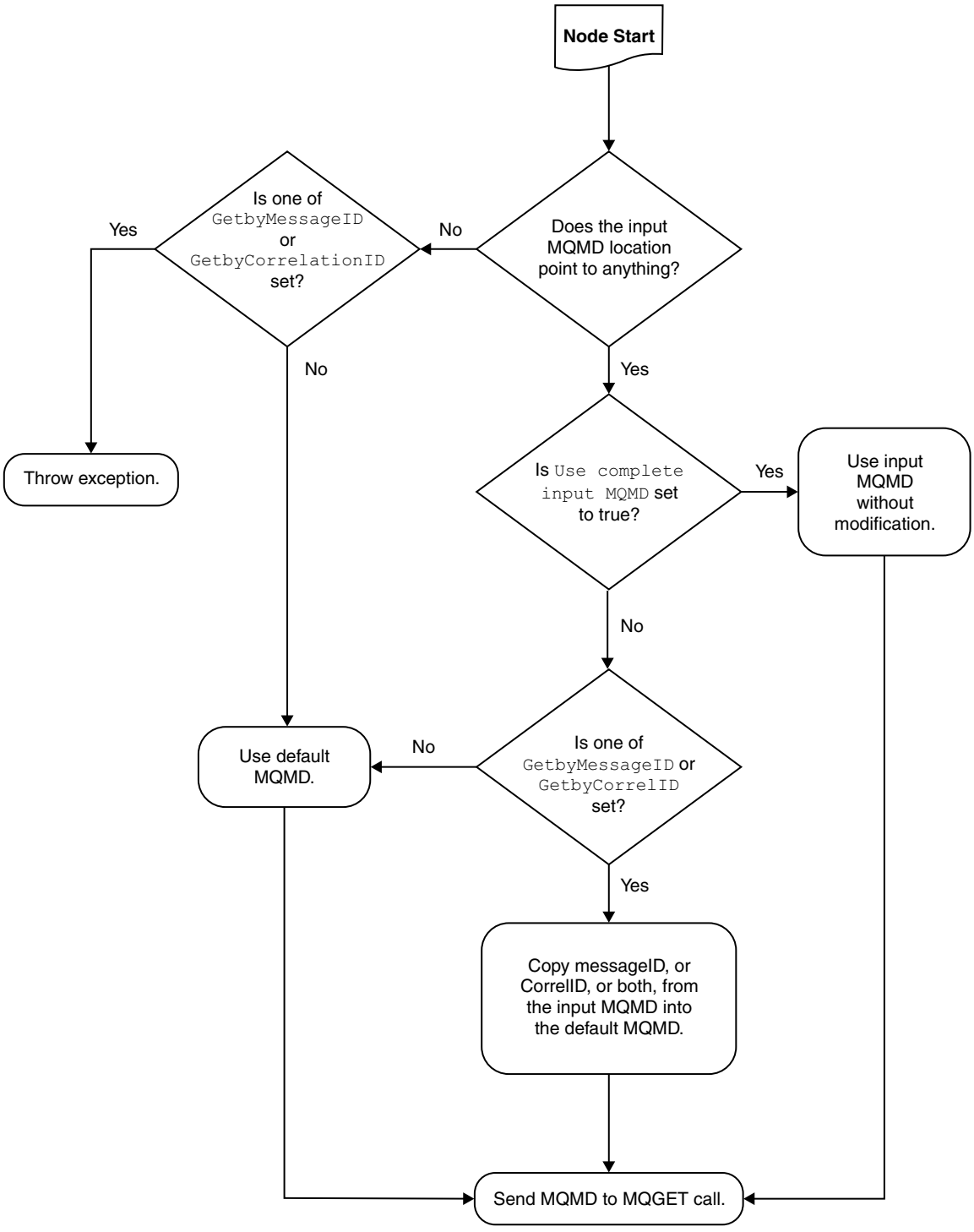
You can parse the MQMD (for example, by using ESQL), where `${outputMQParmsLocation}` is LocalEnvironment.MQ.GET:

```
DECLARE ptr REFERENCE TO OutputLocalEnvironment.MyMQParms;
CREATE FIRSTCHILD OF ptr DOMAIN('MQMD') PARSE(InputLocalEnvironment.MQ.GET.MQMD)
```

How the MQMD for the MQGET call is constructed:

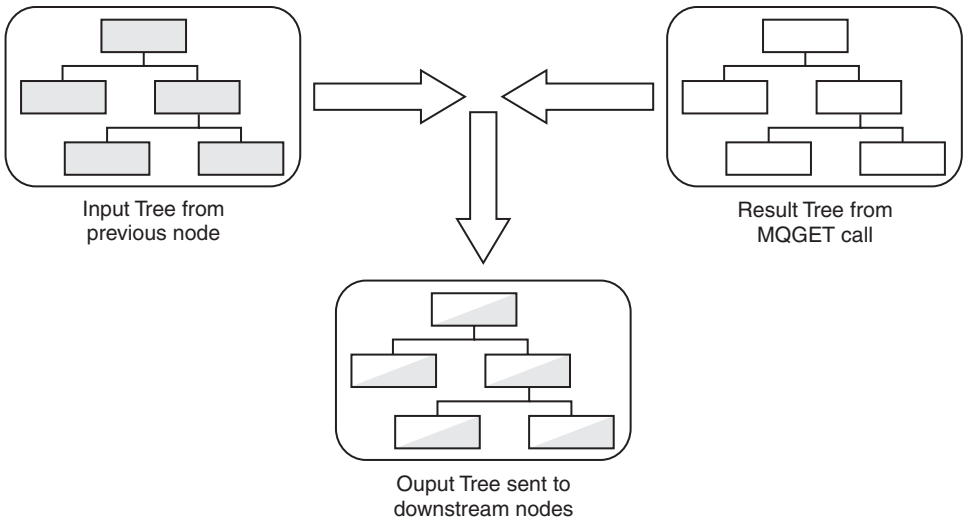
- A default MQMD is prepared. For further information about the MQMD, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.
- If you do not supply an input MQMD, the default MQMD is used.
- If you do supply an input MQMD, the default MQMD is used after the following modifications:
  - If the property Use all input MQMD fields is set, all MQMD fields supplied are copied into the default MQMD from the input MQMD.
  - If the property Use all input MQMD fields is not set and the properties Get by Message ID or Get by Correlation ID are selected, the respective IDs are copied into the default MQMD from the input MQMD.

The following diagram shows how the MQGet node constructs the MQMD that is used on the call to WebSphere MQ:



How the output message tree is constructed:

The following diagram outlines how the MQGet node constructs the output message tree, combining the input tree from the previous node with the result tree from the MQGET call:



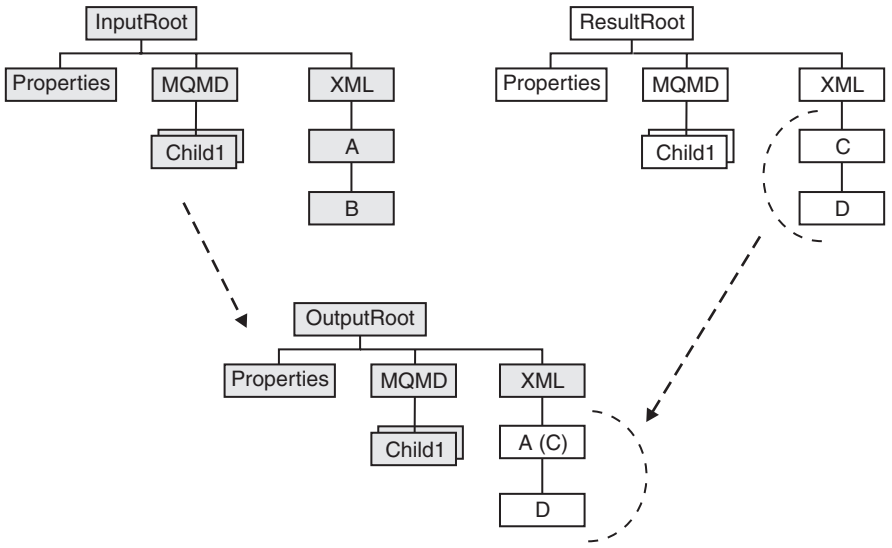
In this example, the MQGet node properties are configured as shown in the following table.

Property	**Action**
Copy Message	Copy Entire Message
Generate Mode	Message
Output Data Location	`OutputRoot.XMLNS.A`
Result Data Location	`ResultRoot.XMLNS.C`

The MQGet node constructs the output tree according to the following sequence:

1. The whole of the input tree is copied to the output tree, including the XML branch with child A, and A's child B.
2. From the result tree, the XML branch's child C, and C's child D, are put into the output tree at position `OutputRoot.XMLNS.A`. Any previous content of A (values and children) is lost, and replaced with C's content, including all values and children it has, in this case child D.
3. The position in the output tree retains the name A.

The following diagram shows this behavior:



For some examples of message trees that are constructed by the MQGet node according to the rules described above, see "MQGet node message tree examples."

MQGet node message tree examples:

The MQGet node generates message trees based on the input message assembly that it receives, and the options that you have set on the node properties.

The message trees, shown in the following table, are generated according to the rules described in "A request-response scenario that uses an MQGet node" on page 230.

With a message assembly like this:	The message that the MQGet node returns is:
InputRoot **MQMD** {input message MQMD} **MQRFH2** {input message MQRFH2} **XMLNS** {input message body} **InputLocalEnvironment** **MQ** **GET** **MQGMO** MatchOptions = MQMO_MATCH_CORREL_ID **MQMD (with no children)** **MyData** **MQMD** {input MQMD} (with CorrelID = {correct Correlation ID as binary})	**ResultRoot** **MQMD** {result message MQMD} **MQRFH2** {result message MQRFH2} **XML** {result message body}

With the following node property settings:	The resulting output message assembly is:
Input MQMD Location InputLocalEnvironment.MyData.MQMD **Copy Message** Copy Entire Message **Copy Local Environment** Copy Entire LocalEnvironment **Generate Mode** Message and LocalEnvironment **Output Data Location** InputLocalEnvironment.MyData.ReturnedMessage	**OutputRoot** **MQMD** {input message MQMD} **MQRFH2** {input message MQRFH2} **XMLNS** {input message body} **OutputLocalEnvironment** **MQ** **GET** **MQGMO** {MQGMO used for MQGET} **MQMD** {MQMD used for MQGET} **CC = 0** **RC = 0** **MyData** **MQMD** {input MQMD} (with CorrelID = {correct Correlation ID as binary}) **ReturnedMessage** **MQMD** {result message MQMD} **MQRFH2** {result message MQRFH2} **XML** {result message body}

With the following node property settings:	The resulting output message assembly is:
Result Data Location ResultRoot.XML	**OutputRoot** **MQMD** {input message MQMD} **MQRFH2** {input message MQRFH2} **XMLNS** {input message body} **OutputLocalEnvironment** **MQ** **GET** **MQGMO** {MQGMO used for MQGET} **MQMD** {MQMD used for MQGET} **CC = 0** **RC = 0** **MyData** **MQMD** {input MQMD} (with CorrelID = {correct Correlation ID as binary}) **ReturnedMessage (with any attributes and value from ResultRoot.XML)** {result message body} This tree is effectively the result of doing an assignment from `${resultDataLocation}` to `${outputDataLocation}`. The value of the source element is copied, as are all children including attributes.

With the following node property settings:	The resulting output message assembly is:
Copy Local Environment None	**OutputRoot** **MQMD** {input message MQMD} **MQRFH2** {input message MQRFH2} **XMLNS** {input message body} **OutputLocalEnvironment** **MQ** **GET** **MQGMO** {MQGMO used for MQGET} **MQMD** {MQMD used for MQGET} **CC = 0** **RC = 0** **MyData** **ReturnedMessage (with any attributes and value from ResultRoot.XML)** {result message body} This tree has the MQMD that is used for the MQGET call in the OutputLocalEnvironment, because the input MQ parameters location had an MQMD element under it. Even though the input tree is not copied, the presence of the MQMD element causes the MQMD that is used for the MQGET call to be placed in the output tree.

With the following node property settings:	The resulting output message assembly is:
Output Data Location <blank> **Copy Local Environment** Copy Entire Local Environment	**OutputRoot** **MQMD** {result message MQMD} **MQRFH2** {result message MQRFH2} **XMLNS** {result message body} **OutputLocalEnvironment** **MQ** **GET** **MQGMO** {MQGMO used for MQGET} **MQMD** {MQMD used for MQGET} **CC = 0** **RC = 0** **MyData** **MQMD** {input MQMD} (with CorrelID = {correct Correlation ID as binary}) The value that you set for the Copy Message property makes no difference to the eventual output tree in this case.

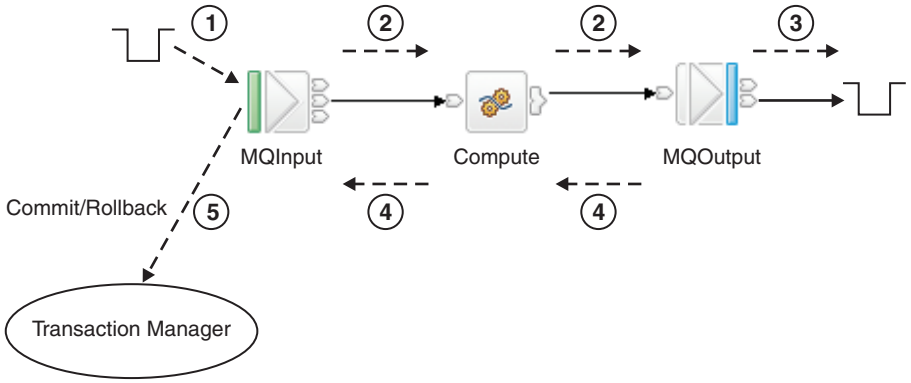
Exploiting user exits

Your message flows can benefit from user exits.

Before you start:

- Read "User exits" on page 167.
- Read Why use a user exit?

The following diagram illustrates how a user exit works. The numbered events are described after the diagram. The MQInput node is used as an example, but the function applies to all input nodes, including user-defined input nodes. Similarly, the Compute and MQOutput nodes could be replaced by any equivalent nodes.



1. (cciInputMessageCallback) The message is dequeued from the input source (read into the flow).

   Built-in nodes and user-defined nodes differ slightly in the way in which user exits are called. For built-in input nodes, the user exit is called as soon as possible after the data has been read from the external source. For user-defined input nodes, the user exit is called just before the node propagates the message.
2. (cciPropagatedMessageCallback) The message is propagated to the node for processing.
3. (cciOutputMessageCallback). A request message is sent to the output node's transport, and transport-specific destination information is written to WrittenDestination in the LocalEnvironment (for example, this information includes the queueName and msgId for an MQ message). The call is made when a node successfully puts a message to a transport, from either an output or a request node. The outputMessageEvent is called by built-in nodes only. The topic for each node that supports WrittenDestination information contains details about the data that it contains.
4. (cciNodeCompletionCallback) Node processing completes.
5. (cciTransactionEventCallback) The user exit is called after the transaction has completed, so that user exit processing is not part of that transaction. The user exit is invoked even if no transactional processing is completed by the flow.

   Where the message flow property Commit Count is greater than one, many-to-one ratios exist between events 1 and 5. This ratio also exists for some scenarios that are specific to the particular input node; for example, when an MQInput node is configured with the Commit by Message Group property selected.

You can write a user exit to track any number of these events. For each of these events, the following data is available to the user exit. All access is read-only, unless stated otherwise:

- The message is dequeued:
  - Bit stream
  - Input node
  - Environment tree (read and write)
- The message is propagated to the node:
  - Message tree (body element read and write)
  - LocalEnvironment tree (read and write)
  - Exception list
  - Environment tree (read and write)
  - Source node

  - Target node
- A message is sent to a transport:
  - Message tree (body element read and write)
  - LocalEnvironment tree (read and write)
  - Exception list
  - Environment tree (read and write)
  - Output or request node
- Node processing completes:
  - Message tree (body element read and write)
  - LocalEnvironment tree (read and write)
  - Exception list
  - Environment tree (read and write)
  - Node
  - Upstream node
  - Exception (if any)
- The end of the transaction:
  - Input node
  - Exception (if any)
  - Environment tree (read and write)

You can register multiple user exits, and, if they are registered, they are invoked in a defined order (see mqsichangeflowuserexits command). Any changes that are made to the message assembly (the message and environment) by a user exit are visible to subsequent user exits.

When the user exit is invoked, it can query the following information:

- Message flow information:
  - Message flow name
  - Broker name
  - Broker's queue manager name
  - Execution group name
  - Message flow's commit count property
  - Message flow's commit interval property
  - Message flow's coordinated transaction property
- Node information:
  - Node name
  - Node type
  - Terminal name
  - Node properties

The user exit can also perform the following tasks:

- Navigate and read the message assembly (Message, LocalEnvironment, ExceptionList, Environment)
- Navigate and write the Message body, LocalEnvironment, and Environment tree

You can register the user exits on a dynamic basis, without needing to redeploy the configuration.

Ensuring that messages are not lost

Messages that flow through your broker domain represent business data that you want to safeguard. Configure the messages, your environment, or both, to ensure that you do not lose messages.

Messages that are generated both by your applications and by runtime components for inter-component communication are important to the operation of your brokers. Messages used internally between components always use the WebSphere MQ protocol. Application messages can use all supported transport protocols.

For application and internal messages traveling across WebSphere MQ, two techniques protect against message loss:

- Message persistence

  If a message is persistent, WebSphere MQ ensures that it is not lost when a failure occurs, by copying it to disk.
- Sync point control

  An application can request that a message is processed in a synchronized unit-of-work (UOW)

For more information about how to use these options, refer to the *System Administration Guide* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.

Internal messages

WebSphere Message Broker components use WebSphere MQ messages to communicate events and data between broker processes and subsystems, and the Configuration Manager and User Name Server. The components ensure that the WebSphere MQ features are used to protect against message loss. You do not need to take any additional steps to configure WebSphere MQ or WebSphere Message Broker to protect against loss of internal messages.

Application messages

If delivery of application messages is critical, you must design application programs and the message flows that they use to ensure that messages are not lost. The techniques used depend on the protocol used by the applications.

WebSphere MQ Enterprise Transport and WebSphere MQ Mobile Transport

If you are using the built-in input nodes that accept messages across the WebSphere MQ or WebSphere MQ Everyplace protocol, you can use the following guidelines and recommendations:

- Using persistent messages

  WebSphere MQ messaging products provide *message persistence*, which defines the longevity of the message in the system and guarantees message integrity. Nonpersistent messages are lost in the event of system or queue manager failure. Persistent messages are always recovered if a failure occurs.

  You can control message persistence in the following ways:

  – Program your applications that put messages to a queue using the MQI or AMI to indicate that the messages are persistent.
  – Define the input queue with message persistence as the default setting.
  – Configure the output node to handle persistent messages.
  – Program your subscriber applications to request message persistence.

  When an input node reads a message from an input queue, the default action is to use the persistence defined in the WebSphere MQ message header (MQMD), that has been set either by the application creating the message, or by the default persistence of the input queue. The message

retains this persistence throughout the message flow, unless it is changed in a subsequent message processing node.

You can override the persistence value of each message when the message flow terminates at an output node. This node has a property that allows you to specify the message persistence of each message when it is put to the output queue, either as the required value, or as a default value. If you specify the default, the message takes the persistence value defined for the queues to which the messages are written.

If a message passes through a Publication node, the persistence of messages sent to subscribers is determined by the registration options of the subscribers. If a subscriber has requested persistent message delivery, and is authorized to do so by explicit or implicit (inherited) ACL, the message is delivered persistently regardless of its existing persistence property. Also, if the user has requested nonpersistent message delivery, the message is delivered nonpersistent regardless of its existing persistence property.

If a message flow creates a message (for example, in a Compute node), the persistence in the MQMD of the new message is copied from the persistence in the MQMD of the incoming message.

- Processing messages under sync point control

  The default action of a message flow is to process incoming messages under sync point in a broker-controlled transaction. This means that a message that fails to be processed for any reason is backed out by the broker. Because it was received under sync point, the failing message is reinstated on the input queue and can be processed again. If the processing fails, the error handling procedures that are in place for this message flow (defined either by how you have configured the message flow, or by the broker) are executed.

  For full details of input node processing, see "Managing errors in the input node" on page 247.

WebSphere MQ Telemetry Transport

If you are using the SCADAInput node that accepts messages from telemetry devices across the MQIsdp protocol, this protocol does not have a concept of queues. Clients connect to a SCADAInput node by specifying the port number on which the node is listening. Messages are sent to clients using a `clientId`. However, you can specify a maximum `QoS` (Quality of Service) within a SCADA subscription message, which is like persistence:

- **`QoS0`** Nonpersistent.
- **`QoS1`** Persistent, but might be delivered more than once
- **`QoS2`** Once and once only delivery

If a persistent SCADA message is published, it might be downgraded to the highest level that the client can accept. In some circumstances, the message might become nonpersistent.

WebSphere MQ Real-time Transport, WebSphere MQ Multicast Transport, and WebSphere MQ Web Services Transport

If you are using the Real-timeInput and Real-timeOptimizedFlow nodes that accept messages from JMS and multicast applications, or you are using the HTTPInput, HTTPRequest, SOAPInput, SOAPRequest nodes, or a SOAPAsyncRequest and SOAPAsyncResponse node pair that accept messages from Web services applications, no facilities are available to

protect against message loss. You can, however, provide recovery procedures by configuring the message flow to handle its own errors.

Other transports and protocols
If you have created your own user-defined input nodes that receive messages from another transport protocol, you must rely on the support provided by that transport protocol, or provide your own recovery procedures.

Providing user-defined properties to control behavior

User-defined properties can be set at design time, deployment time, or run time.

For example, user-defined properties can be queried, discovered, and set at run time to dynamically change the behavior of a message flow. You can use the Configuration Manager Proxy (CMP) to manipulate these properties, which can be used by a systems monitoring tool to perform automated actions in response to situations that it detects in the monitored systems.

- You can use the Message Flow editor to define a user-defined property when you construct a message flow. For more information, see Message Flow editor.
- You can set user-defined properties at deployment time to configure a message flow, as described in "Configuring a message flow at deployment time with user-defined properties" on page 454.
- You can use the CMP API to manipulate user-defined properties on a message flow dynamically at run time, as described in Setting user-defined properties at run time in a CMP application.

Handling errors in message flows

The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

For example, you might design a message flow that expects certain errors that you want to process in a particular way. Or perhaps your flow updates a database, and must roll back those updates if other processing does not complete successfully.

The options that you can use to do this are quite complex in some cases. The options that are provided for MQInput and TimeoutNotification nodes are extensive because these nodes deal with persistent messages and transactions. The MQInput node is also affected by configuration options for WebSphere MQ.

Because you can decide to handle different errors in different ways, there are no fixed procedures to describe. This section provides information about the principles of error handling, and the options that are available, and you must decide what combination of choices that you need in each situation based on the details that are provided in this section.

You can choose one or more of these options in your message flows:

- Connect the Failure terminal of the node to a sequence of nodes that processes the node's internal exception (the fail flow).
- Connect the Catch terminal of the input node or a TryCatch node to a sequence of nodes that processes exceptions that are generated beyond it (the catch flow).

- Insert one or more TryCatch nodes at specific points in the message flow to catch and process exceptions that are generated by the flow connected to the Try terminal.
- Include a Throw node, or code an ESQL THROW statement, to generate an exception.
- Connect the Catch terminal of the AggregateReply node to process aggregation exceptions if you are using aggregation.
- Ensure that all the messages received by an MQInput node are processed in a transaction, or are not processed in a transaction.
- Ensure that all the messages received by an MQInput node are persistent, or are not persistent.

If you include user-defined nodes in your message flow, you must see the information provided with the node to understand how you might handle errors with these nodes. The descriptions in this section cover only the built-in nodes.

When you design your error handling approach, consider the following factors:

- Most of the built-in nodes have Failure terminals. The exceptions are the AggregateControl, AggregateRequest, Input, Label, MQOptimizedFlow, Output, Passthrough, Publication, Real-timeInput, Real-timeOptimizedFlow, Trace, and TryCatch nodes.

  When an exception is detected in a node, the message and the exception information are propagated to the node's Failure terminal. If the node does not have a Failure terminal, or it is not connected, the broker throws an exception and returns control to the closest upstream node that can process it. This node might be a TryCatch node, an AggregateReply node, or the input node.

  If an MQInput node detects an internal error, its behavior is slightly different; if the Failure terminal is not connected, it attempts to put the message to the input queue's backout requeue queue, or (if that is not defined) to the dead letter queue of the broker's queue manager.

  For more information, see “Handling MQInput errors” on page 249.
- Many built-in nodes have Catch terminals:
  - Input nodes: FileInput, HTTPInput, JMSInput, MQInput, PeopleSoftInput, SAPInput, SCADAInput, SiebelInput, SOAPInput, TCPIPClientInput, TCPIPServerInput, TwineballInput
  - Output and response nodes: JMSOutput, SOAPAsyncResponse
  - Routing nodes: AggregateReply
  - Construction nodes: TryCatch
  - Timer nodes: TimeoutNotification

  A message is propagated to a Catch terminal only if it has first been propagated beyond the node (for example, to the nodes connected to the Out terminal).
- When a message is propagated to the Failure or Catch terminal, the node creates and populates a new exception list tree with an exception that represents the error that has occurred. The exception list is propagated as part of the message tree.
- The MQInput and TimeoutNotification nodes have additional processing for transactional messages (other input nodes do not handle transactional messages).

  For more information, see “Handling MQInput errors” on page 249 and “Handling TimeoutNotification errors” on page 252.
- If you include a Trace node that specifies `$Root` or `$Body`, the complete message is parsed. This might generate parser errors that are not otherwise detected.

The general principles of error handling are:

- If you connect the Catch terminal of the input node, you are indicating that the flow handles all the exceptions that are generated anywhere in the out flow. The broker performs no rollback, and takes no action, unless there is an exception on the catch flow. If you want any rollback action after an exception has been raised and caught, you must provide this in the catch flow.
- If you do not connect the Catch terminal of the input node, you can connect the Failure terminal and provide a fail flow to handle exceptions generated by the node. The fail flow is started immediately when an exception occurs in the node.

  The fail flow is also started if an exception is generated beyond the MQInput node (in either out or catch flows), the message is transactional, and the reinstatement of the message on the input queue causes the backout count to reach the backout threshold.

  The HTTPInput and SCADAInput nodes do not propagate the message to the Failure terminal if an exception is generated beyond the node and you have not connected the Catch terminal.
- If a node propagates a message to a catch flow, and another exception occurs that returns control to the same node again, the node handles the message as though the Catch terminal is not connected.
- If you do not connect either the Failure or Catch terminals of the input node, the broker provides default processing (which varies with the type of input node).
- If you need a more comprehensive error and recovery approach, include one or more TryCatch nodes to provide more localized areas of error handling.
- If you have a common procedure for handling particular errors, you might find it appropriate to create a subflow that includes the sequence of nodes required. Include this subflow wherever you need that action to be taken.

For more information, see “Connecting failure terminals” on page 247, “Managing errors in the input node” on page 247, and “Catching exceptions in a TryCatch node” on page 254.

If your message flows include database updates, the way in which you configure the nodes that interact with those databases can also affect the way that errors are handled:

- You can specify whether updates are committed or rolled back. You can set the node property Transaction to specify whether database updates are committed or rolled back with the message flow (option Automatic), or are committed or rolled back when the node itself terminates (option Commit). You must ensure that the combination of these property settings and the message flow error processing give the correct result.
- You can specify how database errors are handled. You can set the properties Treat warnings as errors and Throw exception on database error to change the default behavior of database error handling.

For more information about coordinated database updates, see “Configuring transactionality for message flows” on page 213.

Message flows for aggregation involve additional factors that are not discussed in this topic. For information about message flows for aggregation, see “Handling exceptions in aggregation flows” on page 681.

The following sample demonstrates how to use an error handling routine to trap information about errors and to store that information in a database. The error

handling routine is a subflow that you can add, unchanged, to your message flows. The sample also demonstrates how to configure message flows to control transactionality; in particular, the use of globally coordinated transactions to ensure overall data integrity.

- Error Handler

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

Connecting failure terminals

When a node that has a failure terminal detects an internal error, it propagates the message to that terminal. If it does not have a failure terminal, or if you have not connected the failure terminal, the broker generates an exception.

The nodes sometimes generate errors that you can predict, and it is in these cases that you might want to consider connecting the failure terminal to a sequence of nodes that can take sensible actions in response to the expected errors.

Examples of expected errors are:

- Temporary errors when the input node retrieves the message.
- Validation errors detected by an MQInput, Compute, or Mapping node.
- Messages with an internal or format error that cannot be recognized or processed by the input node.
- Acceptable errors when a node accesses a database, and you choose not to configure the node to handle those errors.
- ESQL errors during message flow development (some ESQL errors cannot be detected by the editor, but are recognized only by the broker; these cause an exception if you have not connected the failure terminal. You can remove the fail flow when you have completely tested the runtime ESQL code).

You can also connect the failure terminal if you do not want WebSphere MQ to try a message again or put it to a backout or dead letter queue.

Managing errors in the input node

When you design your message flow, consider which terminals on the input node to connect to further nodes.

- If the node detects an error, it always propagates the message to the Failure terminal if the node has one, and if you have connected a fail flow.
- If you connect the Catch terminal (if the node has one), this action indicates that you want to handle all exceptions that are generated in the out flow. This method handles errors that can be expected in the out flow. The broker does not take any action unless an exception is generated on the catch flow and the message is transactional. Connect the Failure terminal to handle this case.
- If you do not connect the Catch terminal, or the node does not have a Catch terminal, the broker provides default processing, which depends on the node and whether the message is transactional. Processing for non-transactional messages is described in this topic. Refer to “Handling MQInput errors” on page 249 and “Handling TimeoutNotification errors” on page 252 for details of how these nodes handle transactional messages (other input nodes do not support transactional messages).

All input nodes process non-transactional, non-persistent messages. The built-in input nodes handle failures and exceptions associated with these messages in this way:

- If the node detects an internal error:
  - If you have not connected the Failure terminal, the node logs the error in the local error log and discards the message.

    The Real-timeInput and Real-timeOptimizedFlow nodes retry once before they discard the message; that is, they retrieve the message again and attempt to process it.
  - If you have connected the Failure terminal, you are responsible for handling the error in the fail flow. The broker creates a new ExceptionList to represent the error and this is propagated to the Failure terminal as part of the message tree, but neither the node nor the broker provide any further failure processing.
- If the node has successfully propagated the message to the Out terminal and a later exception results in the message being returned to the input node:
  - If you have not connected the Catch terminal or the node does not have a Catch terminal, the node logs the error in the local error log and discards the message.
  - If you have connected the Catch terminal, you are responsible for handling the error in the catch flow. The broker creates an ExceptionList to represent the error, which is propagated to the Catch terminal as part of the message tree. Neither the node nor the broker provide any further exception processing.
- If the node has already propagated the message to the Catch terminal and an exception is thrown in the catch flow:
  - If you have not connected the Failure terminal, or the input node does not have a Failure terminal, the node logs the error in the local error log and discards the message.
  - If you have connected the Failure terminal, you are responsible for handling the error in the fail flow. The broker creates an ExceptionList to represent the error, which is propagated to the Failure terminal as part of the message tree. Neither the node nor the broker provide any further failure processing.

    The HTTPInput and SCADAInput nodes do not propagate the message to the Failure terminal if an exception is generated in the catch flow. The node logs the error in the local error log and discards the message.
- If the node has propagated the message to the Failure terminal and an exception is thrown in the fail flow, the node logs the error in the local error log and discards the message.

In every situation in which it discards the message, the HTTPInput node waits until the time specified by the node property Maximum client wait time expires, and returns an error to the Web services client.

This action is summarized in the following table.

Error event	**Failure terminal connected**	**Failure terminal not connected**	**Catch terminal connected**	**Catch terminal not connected**
Node detects internal error	Fail flow handles the error	Node logs the error and discards the message	Not applicable	Not applicable

Error event	Failure terminal connected	Failure terminal not connected	Catch terminal connected	Catch terminal not connected
Node propagates message to Out terminal, exception occurs in out flow	Not applicable	Not applicable	Catch flow handles the error	Node logs the error and discards the message
Node propagates message to Catch terminal, exception occurs in catch flow	Fail flow handles the error (not HTTPInput or SCADAInput)	Node logs the error and discards the message	Not applicable	Not applicable
Node propagates message to Failure terminal, exception occurs in fail flow	Not applicable	Not applicable	Node logs the error and discards the message	Node logs the error and discards the message

Handling MQInput errors:

The MQInput node takes certain actions when it handles errors with persistent and transactional messages. The node attempts retry processing when a message is rolled back to the input queue.

Errors encountered with non-transactional messages are handled as described in "Managing errors in the input node" on page 247.

- The MQInput node detects an internal error in the following situations:
  - A message validation error occurs when the associated message parser is initialized.
  - A warning is received on an MQGET call.
  - The backout threshold is reached when the message is rolled back to the input queue.
- If the MQInput node detects an internal error, one of the following actions occur:
  - If you have not connected the Failure terminal, the MQInput node attempts to put the message to the input queue's backout requeue queue, or (if that is not defined) to the dead letter queue of the broker's queue manager. If the put attempt fails, the message is rolled back to the input queue. The MQInput node writes the original error and the MQPUT error to the local error log. The MQInput node now invokes the retry logic, described in "Retry processing" on page 250.
  - If you have connected the Failure terminal, you are responsible for handling the error in the flow connected to the Failure terminal. The broker creates a new exception list to represent the error and this is propagated to the Failure terminal as part of the message tree, but neither the MQInput node nor the broker provide any further failure processing.
- If the MQInput node has successfully propagated the message to the out terminal and an exception is thrown in the out flow, the message is returned to the MQInput node:
  - If you have not connected the Catch terminal, the message is rolled back to the input queue. The MQInput node writes the error to the local error log and invokes the retry logic, described in "Retry processing" on page 250.

  - If you have connected the Catch terminal, you are responsible for handling the error in the flow connected to the Catch terminal. The broker creates a new exception list to represent the error and this list is propagated to the Catch terminal as part of the message tree, but neither the MQInput node nor the broker provide any further failure processing.
- If the MQInput node has already propagated the message to the Catch terminal and an exception is thrown in the flow connected to the Catch terminal, the message is returned to the MQInput node:
  - The MQInput node writes the error to the local error log.
  - The message is rolled back to the input queue.
- If the MQInput node has already propagated the message to the Failure terminal and an exception is thrown in the flow connected to the Failure terminal, the message is returned to the MQInput node and rolled back to the input queue. The MQInput node writes the error to the local error log and invokes the retry logic, described in "Retry processing." The message is not propagated to the Catch terminal, even if that is connected.

This action is summarized in the following table.

Error event	Failure terminal connected	Failure terminal not connected	Catch terminal connected	Catch terminal not connected
Node detects internal error	Flow connected to the Failure terminal handles the error	Message put to alternative queue; node retries if the put fails	Not applicable	Not applicable
Node propagates message to out terminal, exception occurs in out flow	Not applicable	Not applicable	Flow connected to the Catch terminal handles the error	Node retries
Node propagates message to Catch terminal, exception occurs in flow connected to the Catch terminal	Error logged, message rolled back	Error logged, message rolled back	Not applicable	Not applicable
Node propagates message to Failure terminal, exception occurs in flow connected to the Failure terminal	Not applicable	Not applicable	Node retries	Node retries

Retry processing:

The node attempts retry processing when a message is rolled back to the input queue. It checks whether the message has been backed out before, and if it has, whether the backout count has reached (equalled) the backout threshold. The backout count for each message is maintained by WebSphere MQ in the MQMD.

You specify (or allow to default to 0) the backout threshold attribute *BOTHRESH* when you create the queue. If you accept the default value of 0, the node increases

this to 1. The node also sets the value to 1 if it cannot detect the current value. This means that if a message has not been backed out before, it is backed out and retried at least once.

1. If the node has propagated a message to the out terminal many times following repeated failed attempts in the out flow, and the number of retries has reached the backout threshold limit, it attempts to propagate the message through the Failure terminal if that is connected. If you have not connected the Failure terminal, the node attempts to put the message to another queue.

   If a failure occurs beyond the Failure terminal, further retries are made until the backout count field in the MQMD reaches twice the backout threshold set for the input queue. When this limit is reached, the node attempts to put the message to another queue.
2. If the backout threshold has not been reached, the node gets the message from the queue again. If this fails, this is handled as an internal error (described above). If it succeeds, the node propagates the message to the out flow.
3. If the backout threshold has been reached:
   - If you have connected the Failure terminal, node propagates the message to that terminal. You must handle the error on the flow connected to the Failure terminal.
   - If you have not connected the Failure terminal, the node attempts to put the message on an available queue, in order of preference:
     a. The message is put on the input queue's backout requeue name (queue attribute *BOQNAME*), if one is defined.
     b. If the backout queue is not defined, or it cannot be identified by the node, the message is put on the dead letter queue (DLQ), if one is defined. (If the broker's queue manager has been defined by the **mqsicreatebroker** command, a DLQ with a default name of SYSTEM.DEAD.LETTER.QUEUE has been defined and is enabled for this queue manager.)
     c. If the message cannot be put on either of these queues because there is an MQPUT error (including queue does not exist), or because they cannot be identified by the node, it cannot be handled safely without risk of loss.

        The message cannot be discarded, therefore the message flow continues to attempt to backout the message. It records the error situation by writing errors to the local error log. A second indication of this error is the continual incrementing of the *BackoutCount* of the message in the input queue.

        If this situation has occurred because neither queue exists, you can define one of the backout queues mentioned above. If the condition preventing the message from being processed has cleared, you can temporarily increase the value of the *BOTHRESH* attribute. This forces the message through normal processing.
4. If twice the backout threshold has been reached or exceeded, the node attempts to put the message on an available queue, in order of preference, as defined in the previous step.

Handling message group errors:

WebSphere MQ supports message groups. You can specify that a message belongs to a group and its processing is then completed with reference to the other messages in the group (that is, either all messages are committed or all messages

are rolled back). When you send grouped messages to a broker, this condition is upheld if you have configured the message flow correctly, and errors do not occur during group message processing.

To configure the message flow to handle grouped messages correctly, follow the actions described in the "MQInput node" on page 1083. However, correct processing of the message group cannot be guaranteed if an error occurs while one of the messages is being processed.

If you have configured the MQInput node as described, under normal circumstances all messages in the group are processed in a single unit of work which is committed when the last message in the group has been successfully processed. However, if an error occurs before the last message in the group is processed, the unit of work that includes the messages up to and including the message that generates the error is subject to the error handling defined by the rules documented here, which might result in the unit of work being backed out.

However, any of the remaining messages in the group might be successfully read and processed by the message flow, and therefore are handled and committed in a new unit of work. A commit is issued when the last message is encountered and processed. Therefore if an error occurs in a group, but not on the first or last message, it is possible that part of the group is backed out and another part committed.

If your message processing requirements demand that this situation is handled in a particular way, you must provide additional error handling to handle errors in message groups. For example, you could record the failure of the message group in a database, and include a check on the database when you retrieve each message, forcing a rollback if the current group has already encountered an error. This would ensure that the whole group of messages is backed out and not processed unless all are successful.

Handling TimeoutNotification errors:

The TimeoutNotification node takes various actions when it handles errors with transactional messages, depending on whether the Failure and Catch terminals are connected.

Errors that are encountered with non-transactional messages are handled as described in "Managing errors in the input node" on page 247.

- If the TimeoutNotification node detects an internal error, one of the following actions occur:
  - If you have not connected the Failure terminal:
    1. The TimeoutNotification node writes the error to the local error log.
    2. The TimeoutNotification node repeatedly tries to process the request until the problem has been resolved.
  - If you have connected the Failure terminal, you are responsible for handling the error in the flow connected to the Failure terminal. The broker creates an ExceptionList to represent the error, which is propagated to the Failure terminal as part of the message tree. Neither the TimeoutNotification node nor the broker provide any further failure processing. The message is written to the Failure terminal as part of the same transaction, and if the failure flow handles the error successfully the transaction is committed.
- If the TimeoutNotification node has successfully propagated the message to the Out terminal and an exception is thrown in the flow connected to the Out

terminal, the message is returned to the TimeoutNotification node. The TimeoutNotification node writes the error to the local error log and does one of the following:

  – If you have not connected the Catch terminal, the TimeoutNotification node tries to process the message again until the problem is resolved.
  – If you have connected the Catch terminal, you are responsible for handling the error in the flow connected to the Catch terminal. The broker creates an ExceptionList to represent the error, which is propagated to the Catch terminal as part of the message tree. Neither the TimeoutNotification node nor the broker provide any further failure processing. The message is written to the Catch terminal as part of the same transaction, and if the flow connected to the Catch terminal handles the error successfully the transaction is committed.

- If the TimeoutNotification node has already propagated the message to the Catch terminal and an exception is thrown in the flow connected to the Catch terminal, the message is returned to the TimeoutNotification node. The TimeoutNotification node writes the error to the local error log and tries to process the message again.
- If the TimeoutNotification node has already propagated the message to the Failure terminal and an exception is thrown in the flow connected to the Failure terminal, the message is returned to the TimeoutNotification node. The TimeoutNotification node writes the error to the local error log, and tries to process the message again. The message is not propagated to the Catch terminal, even if that is connected.

This action is summarized in the following table.

Error event	**Failure terminal connected**	**Failure terminal not connected**	**Catch terminal connected**	**Catch terminal not connected**
Node detects internal error	Flow connected to the Failure terminal handles the error	Error logged, node retries	Not applicable	Not applicable
Node propagates message to out terminal, exception occurs in out flow	Not applicable	Not applicable	Flow connected to the Catch terminal handles the error	Error logged, node retries
Node propagates message to Catch terminal, exception occurs in flow connected to the Catch terminal	Error logged, node retries	Error logged, node retries	Not applicable	Not applicable
Node propagates message to Failure terminal, exception occurs in flow connected to the Failure terminal	Not applicable	Not applicable	Error logged, node retries	Error logged, node retries

Catching exceptions in a TryCatch node

You can design a message flow to catch exceptions before they are returned to the input node. You can include one or more TryCatch nodes in a flow to provide a single point of failure for a sequence of nodes. With this technique, you can provide very specific error processing and recovery.

A TryCatch node does not process a message in any way, it represents only a decision point in a message flow. When the TryCatch node receives a message, it propagates it to the Try terminal. The broker passes control to the sequence of nodes connected to that terminal (the try flow).

If an exception is thrown in the try flow, the broker returns control to the TryCatch node. The node writes the current contents of the exception list tree to the local error log, then writes the information for the current exception to the exception list tree, overwriting the information that is stored there.

The node propagates the message to the sequence of nodes connected to the Catch terminal (the catch flow). The content of the message tree that is propagated is identical to the content that was propagated to the Try terminal, which is the content of the tree when the TryCatch node first received it. The node enhances the message tree with the new exception information that it has written to the exception list tree. Any modifications or additions that the nodes in try flow made to the message tree are not present in the message tree that is propagated to the catch flow.

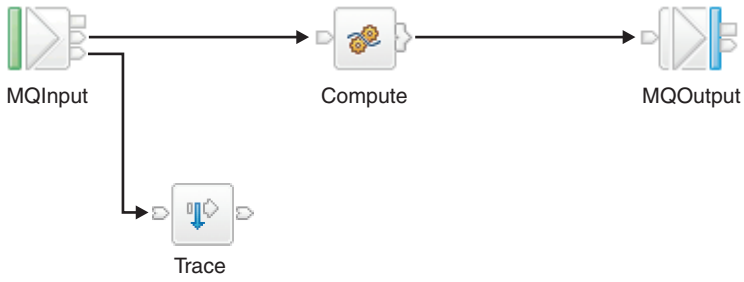
However, if the try flow has completed processing that involves updates to external databases, these updates are not lost; they persist while the message is processed by the catch flow, and the decision about whether the updates are committed or rolled back is made on the configuration of your message flow and the individual nodes that interact with the databases. If the updates are committed because of the configuration that you have set, you must include logic in your catch flow that rolls back the changes that were made.

To review the options for configuration, see “Configuring transactionality for message flows” on page 213.

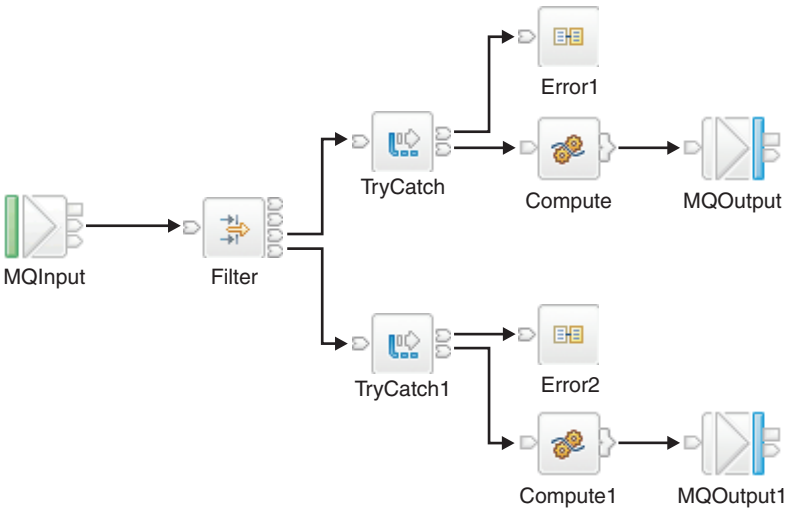
The broker returns control to the next catch point in the message flow (which might be another TryCatch node, but is always, in the last case, the input node) if one of the following conditions is true:

- An exception is thrown in the catch flow of the TryCatch node (for example, if you include a Throw node, or code an ESQL THROW statement, or if the broker generates the exception).
- You do not connect the Catch terminal of the TryCatch node.

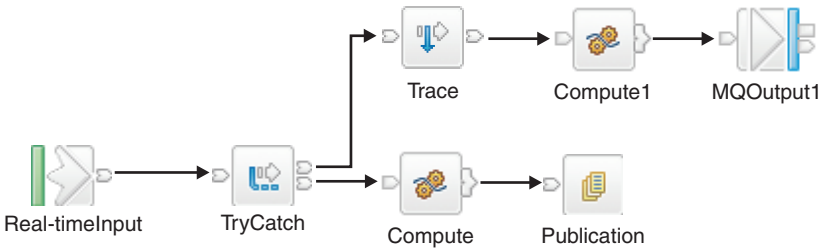
The following example shows how you can configure the flow to catch exceptions in the input node. The MQInput node's Catch terminal is connected to a Trace node to record the error.



In the following example, the message flow has two separate processing flows connected to the Filter node's True and False terminals. Here a TryCatch node is included on each of the two routes that the message can take. The Catch terminal of both TryCatch nodes is connected to a common error processing subflow.



If the input node in your message flow does not have a Catch terminal (for example, Real-timeInput), and you want to process errors in the flow, you must include a TryCatch node. The following example shows how you could connect a flow to provide this error processing. In this example, you could configure the ESQL in the Compute node on the catch flow to examine the exception that has been caught and set the output queue name dynamically.



Managing message flows

Manage your message flows and associated resources in the workbench.

- "Creating a message flow project"
- "Deleting a message flow project" on page 257
- "Creating a broker schema" on page 258
- "Creating a message flow" on page 259
- "Opening an existing message flow" on page 261
- "Copying a message flow by using copy" on page 261
- "Renaming a message flow" on page 262
- "Moving a message flow" on page 263
- "Deleting a message flow" on page 264
- "Deleting a broker schema" on page 265
- "Version and keyword information for deployable objects" on page 265
- "Saving a message flow" on page 266

To learn more about message flows look at the following sample:

- Airline Reservations

In this sample, you can explore message flow resources, and learn how to create, delete, and rename the resources.

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

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

Creating a message flow project

A message flow project is a container for message flows; you must create a project before you can create a message flow.

Before you start:

Read the concept topic about message flow projects.

The project and its resources are stored in a file system or in a shared repository. If you are using a file system, you can use the local file system or a shared drive. If you store files in a repository, you can use all the available repositories that are supported by Eclipse; for example, CVS.

To create a message flow project and other resource files that you need to start developing applications, you can use a Quick Start wizard (as described in "Quick Start wizards overview" on page 169).

To create only a message flow project, perform the following actions:

1. Switch to the Broker Application Development perspective.
2. Click **File** → **New** → **Message Flow Project** or right-click any resource in the Broker Development view and click **New** → **Message Flow Project**.

   You can also press Ctrl+N to display a dialog box, from which you select the wizard to create a new object. Click **Message Brokers** in the left view; the right

view displays a list of objects that you can create for WebSphere Message Broker. Click **Message Flow Project** in the right view, then click **Next**. The New Message Flow Project wizard opens.

3. Enter a name for the project. Choose a project name that reflects the message flows that it contains. For example, if you want to use this project for financial processing message flows, you might give it the name Finance_Flows.
4. To use the default location for the new message project directory (the `\workspace` subdirectory of your current installation), leave the **Use default** check box selected (it is selected when the dialog box opens). When you choose this option, you cannot edit the **Directory** field.

   If you do not want to use the default location, clear the **Use default** check box and specify a location for the new message flow project files in **Directory**.
5. If this message flow project depends on other message flow projects or message set projects, click **Next**.

   A list of current projects is displayed. Select one or more message flow projects, one or more message set projects, or both, from the list to indicate the dependencies of this new message flow project. Message flow projects and message set projects are filtered to show only resources in the active working set.

   This message flow project depends on another message flow project if you intend to use common resources in it. Common resources that you can share between message flow projects are:
   a. ESQL subroutines (defined in broker schemas)
   b. Mappings
   c. Message sets
   d. Subflows

   For example, you might want to reuse a subflow that provides standard error processing, such as writing the message to a database, or recording a trace entry.

   This message flow project depends on a message set project if you intend to refer to the message it defines in ESQL in the message flow nodes.

   You can add dependencies after you have created the message flow project by right-clicking the project in the Broker Development view and clicking **Properties**. Click **References** and select the dependent message flow or message set project from the list of projects displayed.
6. Click **Finish** to complete the task.

The project file is created in a directory that has the same name as your message flow project in the specified location. All other files that you create that are related to this message flow project are created in this same directory.

A default broker schema (`default`) is also created in the project. You can create and use different schemas in a single project to organize message flow resources and to provide the scope of resource names to ensure uniqueness.

Next: create a message flow.

Deleting a message flow project

A message flow project is the container in which you create and maintain all the resources associated with one or more message flows. These resources are created as files, and are displayed in the project in the Broker Development view. If you do not want to retain a message flow project, you can delete it.

Before you start:

- Create a message flow project
- Read the concept topic about message flow projects

When you delete a message flow project in the workbench, you delete the project and its resources, and the Configuration Manager destroys its copy. If you are using a shared repository, the repository might retain a copy of a deleted resource.

To delete a message flow project:

1. Switch to the Broker Application Development perspective.
2. Select the message flow project that you want to delete, and click **Edit** → **Delete**. You can also press Delete, or right-click the project in the Broker Development view and click **Delete**.
3. You must choose if you want the contents of the message flow project folder to be deleted with this action in the displayed confirmation dialog box. The dialog box contains two buttons:
   a. The first button confirms that all contents are to be deleted.
   b. The second button requests that the directory contents are not deleted. The default action is not to delete the contents; therefore, the second button is selected by default when the dialog box is initially displayed.
   a. Select the appropriate button. If you choose not to delete the contents of the message flow project directory, all the files and the directory itself are retained.

      If you later create another project with the same name, and specify the same location for the project (or accept the location as the default value), you can access the files previously created.

      If you choose to delete all the contents, all files and the directory itself are deleted.
4. Click **Yes** to complete the delete request, or **No** to terminate the delete request.

When you click **Yes**, the requested objects are deleted.

If you maintain resources in a shared repository, a copy is retained in that repository. You can follow the instructions provided by the repository supplier to retrieve the resource, if required.

If you are using the local drive or a shared drive to store your resources, no copy of the resource is retained. Be careful to select the correct resource when you complete this task.

Creating a broker schema

To organize your message flow project resources, and to define the scope of resource names to ensure uniqueness, you can create broker schemas. A default schema is created when you create the message flow project, but you can create additional schemas.

Before you start:

- Create a message flow project
- Read the concept topic about broker schemas

To create a broker schema:

1. Switch to the Broker Application Development perspective.

2. Click **File** → **New** → **Broker Schema** or right-click any resource in the Broker Development view and click **New** → **Broker Schema**.

   You can also press Ctrl+N. This displays a dialog that allows you to select the wizard to create a new object. Click Message Brokers in the left view. The right view displays a list of objects that you can create for WebSphere Message Broker. Click **Broker Schema** in the right view, then click **Next**. The New Broker Schema wizard displays.

3. Enter the message flow project in which you want the new schema to be created. If a message flow project or one of its resources is highlighted when you start the wizard, that project name appears in the dialog box. If a name does not appear in this field, or if you want to create the schema in another project, click **Browse** and select the correct project from the displayed list. The message flow project list is filtered to show projects in the active working set.

   You can type the project name in, but you must enter a valid name. The dialog box displays a red cross and the error message `The specified project does not exist` if your entry is not a valid project. You must specify a message flow project; if you select a message set project, you cannot complete the operation.

4. Enter a name for the schema. Choose a name that reflects the resources that it contains. For example, if you want to use this schema for message flows for retail applications, you might give it the name `Retail`.

   A broker schema name must be a character string that starts with a Unicode character followed by zero or more Unicode characters or digits, and the underscore symbol (_). You can use the period to provide a structure to the name, for example `Stock.Common`.

5. Click **Finish** to complete the task.

The schema directory is created in the project directory. If the schema name is structured by using periods, further subdirectories are defined. For example, the broker schema `Stock.Common` results in a directory `Common`, in directory `Stock`, in the message flow project directory.

Creating a message flow

Create a message flow to specify how to process messages in the broker. You can create one or more message flows and deploy them to one or more brokers.

Before you start:

- Complete the following task: “Creating a message flow project” on page 256.
- Read the concept topic about “Broker schemas” on page 139.

The mode that your broker is working in can affect the number of message flows that you can use; see Restrictions that apply in each operation mode.

The message flow and its resources are stored in a file system or in a shared repository. If you are using a file system, it can be on the local drive, or a shared drive. If you store files in a repository, you can use all the available repositories that are supported by Eclipse; for example, CVS.

Use this process to create a complete message flow that you can deploy, or a subflow that provides a subset of function (for example, a reusable error processing routine), that you cannot deploy on its own.

To create a message flow and other resource files that you need to start developing applications, you can use a Quick Start wizard.

To create only a message flow, perform the following actions:

1. Switch to the Broker Application Development perspective.
2. If you have not already created the message flow project in which you want to create the message flow, you can either create it now, or you can create the message flow project as an optional step in creating the message flow (in step 4). If you want to create the message project first, see "Creating a message flow project" on page 256. The project can be empty, or can have message flows defined in it.
3. Complete one of the following actions to open a message flow:
   - Click **File** → **New** → **Message Flow**.
   - Right-click a resource in the Broker Development view and click **New** → **Message Flow**.
   - Press Ctrl+N. This action displays a dialog box in which you can select the wizard to create an object:
     a. Click **Message Brokers** in the left view. The right view displays a list of objects that you can create for WebSphere Message Broker.
     b. Click **Message Flow** in the right view, then click **Next**. The New Message Flow wizard displays.
4. Identify the project in which you want to define the message flow. This field is filtered to show only resources in the active working set.
   - If you have a resource selected in the Broker Development view, the name of the corresponding project is displayed in the Message Flow Project field.
   - If you do not have a resource selected, the first field is blank.
     – If you have already created the message flow project for this message flow, you can perform either of the following actions:
       - Type the name of the project into the field.
       - Click the down-arrow and select the appropriate project from the list displayed.
     – If you have not already created the message flow project, select **New**. The New Message Flow Project wizard starts, and you can use it to create the message flow project for your new message flow, as described in "Creating a message flow project" on page 256.

       When you have finished creating the message flow project, the New Message Flow Project wizard closes, and the name of your new message flow project is displayed in the Message Flow Project field of the New Message Flow window.

   If your entry is not a valid project name, the window displays a red cross, and the error message `The specified project does not exist`.
5. In the Message flow **Name** field, enter the name of the new message flow. You can use all valid characters for the name, but it is helpful to choose a name that reflects its function, for example, OrderProcessing.
6. Decide whether you want to use the default broker schema. When you create a message flow project, a default schema is created in it, and this default value is assumed unless you clear it. You can create and use different schemas in a single project to organize message flow resources, and to provide the scope of resource names to ensure uniqueness.
   - If you want the message flow to be created in the default broker schema, ensure that you select **Use default** in the Flow organization section.
   - If you want to use a different broker schema, clear **Use default**. You can now perform either of the following actions:

- Enter the name of the broker schema into the **Schema** field.
- Click **Browse** to select from the broker schemas in the message flow project.

7. Click **Finish**.

The new message flow (`<message_flow_name>.msgflow`) is displayed in its project in the Broker Development view. The Editor view is empty, and ready to populate.

Next, you can do either of the following tasks:

- "Saving a message flow" on page 266
- "Defining message flow content" on page 268

Opening an existing message flow

Open an existing message flow to change or update its contents, or to add or remove nodes.

Before you start

You must have completed the following task:

- "Creating a message flow" on page 259

To open an existing message flow:

1. Switch to the Broker Application Development perspective. The Broker Development view is populated with all the message flow and message set projects that you have access to. A message flow is contained in a file called `<message_flow_name>.msgflow`.
2. Right-click the message flow that you want to work with, and click **Open**. Alternatively you can double-click the message flow in the Broker Development view.

   The graphical view of the message flow is displayed in the editor view. You can now work with this message flow; for example, you can add or remove nodes, change connections between nodes, or modify node properties.
3. Click **Open ESQL** for any node in the flow that requires ESQL, or double-click the ESQL file (the .esql file) in the Broker Development view to open it, if you want to work with the ESQL file for this message flow.
4. Click **Open Mappings** for any node in the flow that requires mappings, or double-click the mappings file (the `.msgmap` file) in the Broker Development view to open it, if you want to work with the mappings file for this message flow.
5. Click **Open Java** for any JavaCompute node in the flow, or double-click the Java file in the Broker Development view to open it, if you want to work with the Java file for this message flow.

Copying a message flow by using copy

You might find it useful to copy a message flow as a starting point for a new message flow that has similar function. For example, you might want to replace or remove one or two nodes to process messages in a different way.

Before you start:

To complete this task, you must have created a message flow, as described in "Creating a message flow" on page 259.

To copy a message flow:

1. Switch to the Broker Application Development perspective.
2. Select the message flow (<message_flow_name>.msgflow) that you want to copy in the Broker Development view.
   a. Right-click the file and click **Copy** from the menu.
3. Right-click the broker schema in the message flow project to which you want to copy the message flow and click **Paste**. You can copy the message flow in the same broker schema in the same message flow, or to a different broker schema in the same message flow project, or to a broker schema in a different message flow project.

   When you copy a message flow, the associated files (ESQL and mapping, if present) are not automatically copied to the same target message flow project. If you want these files copied as well, you must do this explicitly following this procedure.

   You might also need to update nodes that have associated ESQL or mappings, to ensure that modules are unique.

   For example, if you have created a message flow Test1 that contains a single Compute node, and you copy message flow Test1 and its associated `.esql` file to the same broker schema in the same message flow project (and give the new copy a different name, for example Test2), two modules named Test1_Compute now exist in the single schema. One is in `Test1.esql`, the second in `Test2.esql`.

   This duplication is not supported, and an error message is written to the Problems view when you have completed the copy action. You must rename the associated ESQL modules in the `.esql` file and update the matching node properties to ensure that every module in a broker schema is unique.

The message flow is copied with all property settings intact. If you intend to use this copy of the message flow for another purpose, for example to retrieve messages from a different input queue, you might have to modify its properties.

You can also use **File → Save As** to copy a message flow. This task is described in "Saving a message flow" on page 266.

Renaming a message flow

You can rename a message flow. You might want to rename a flow if you have modified the message flow to provide a different function, and you want the name of the message flow to reflect this new function.

Before you start

To complete this task, you must have completed the following task:

- "Creating a message flow" on page 259

To rename a message flow:

1. Switch to the Broker Application Development perspective.
2. Right-click the message flow that you want to rename (<message_flow_name>.msgflow) in the Broker Development view, and click **Rename**, or click **File → Rename**. If you have the message flow selected, you can also press F2. The Rename Resource dialog is displayed.
3. Type in the new name for the message flow.
4. Click **OK** to complete this action, or **Cancel** to cancel the request. If you click **OK**, the message flow is renamed.

After you have renamed the message flow, any references that you have to this message flow (for example, if it is embedded in another message flow) are no longer valid.

5. You must open the affected message flows and correct the references if you are not sure where you have embedded this message flow.
   a. Click **File** → **Save All** The save action saves and validates all resources. Unresolved references are displayed in the Problems view; click each error listed to open the message flow that makes a non-valid reference in the editor view.
   b. Right-click the subflow icon and click **Locate Subflow**. The Locate Subflow dialog is displayed, listing the available message flow projects.
   c. Expand the list and explore the resources available to locate the required subflow.
   d. Select the correct subflow, and click **OK**. All references in the current message flow are updated for you. All related errors are removed from the Problems view.

Moving a message flow

You can move a message flow from one broker schema to another in the same project or to a broker schema in another project. You might want to move a flow, for example, if you are reorganizing the resources in your projects.

Before you start:

Complete the following task:

- "Creating a message flow" on page 259

To move a message flow:

1. Switch to the Broker Application Development perspective.
2. Drag the message flow that you want to move from its current location to a broker schema in the same or another message flow project.

   If the target location that you have chosen is not valid, a black no-entry icon appears over the target, an error dialog box is displayed, and the message flow is not moved. You can move a message flow to another schema in the same project or to a schema in another message flow project.

   Alternatively, you can use the following method:
   a. Right-click the message flow that you want to move (*message_flow_name*.msgflow) in the Broker Development view and click **Move**, or **File** → **Move**. The Move dialog box is displayed, and contains a list of all valid projects to which you can move this message flow.
   b. Select the project and the broker schema in the project to which you want to move the message flow. You can move a message flow to another schema in the same project or to a schema in another message flow project.
   c. Click **OK** to complete the move, or **Cancel** to cancel the move. If you click **OK**, the message flow is moved to its new location.
3. Check the Problems view for errors or warnings that are generated by the move. Errors are indicated by the error icon  , warnings are indicated by the warning icon  . The errors in this view include those that are caused by broker references. When the move is complete, all references to this message flow (for example, if this message flow is a reusable error routine that you have embedded in another message flow) are checked.

If you have moved the message flow in the same broker schema (in the same or another project), all references are still valid. However, if you move the message flow from one broker schema to another (in the same or a different project), the references are broken because the resources are linked by a fully-qualified name of which the broker schema is a part. Information about broken references is written to the Problems view; for example, `Linked or nested flow mflow1 cannot be located`.

4. Double-click each error or warning to correct it. The message flow that contains the error is opened in the editor view and the node in error is highlighted.

When you move a message flow, the associated files (for example, all ESQL or mapping files) are not automatically moved to the target broker schema. If you want to move these files as well, you must do so explicitly by following the procedure in this topic.

Deleting a message flow

Delete message flows from your message flow project when you no longer need them.

Deleting a message flow in the workbench deletes the project and its resources, and the Configuration Manager destroys its copy. If you are using a shared repository, the repository might retain a copy of a deleted resource.

Before you start:

Complete the following task:

- "Creating a message flow" on page 259

To delete a message flow:

1. Switch to the Broker Application Development perspective.
2. Select the message flow in the Broker Development view (`<message_flow_name>.msgflow`), and press the Delete key. A confirmation dialog is displayed.

   Alternatively, right-click the message flow in the Broker Development view and click **Delete**, or click **Edit → Delete**. The same dialog is displayed.
3. Click **Yes** to delete the message flow definition file, or **No** to cancel the delete request. When you click **Yes**, the requested objects are deleted.

   If you maintain resources in a shared repository, a copy is retained in that repository. You can follow the instructions provided by the repository supplier to retrieve the resource if required.

   If you are using the local file system or a shared file system to store your resources, no copy of the resource is retained. Be careful to select the correct resource when you complete this task.
4. Check the Problems view to see if errors have been returned to the delete request. Errors are generated if you delete a message flow that is embedded in another flow, because the reference is no longer valid.
   a. Click the error in the Problems view. The message flow that now has a non-valid reference is displayed.
   b. Either remove the node that represents the deleted message flow from the parent message flow, or create another message flow with the same name to provide whatever processing is required.

When you delete the message flow, the files that are associated with the message flow (the ESQL and mapping files, if present) are not deleted by this action. If you want to delete these files also, you must do so explicitly.

Deleting a broker schema

You can delete a broker schema that you have created in a message flow project if you no longer need it.

Before you start:

- Create a broker schema
- Read the concept topic about broker schemas

To delete a broker schema:

1. Switch to the Broker Application Development perspective.
2. Select the broker schema in the Broker Development view and press the Delete key. A confirmation dialog box is displayed.

   You can also right-click the broker schema in the Broker Development view and click **Delete**, or click **Edit → Delete**. The same dialog box is displayed.

   If the broker schema contains resources, the **Delete** menu option is disabled, and the Delete key has no effect. You must delete all resources in the schema before you can delete the schema.
3. Click **Yes** to delete the broker schema directory or **No** to cancel the delete request. When you click **Yes**, the requested objects are deleted.

   If you maintain resources in a shared repository, a copy is retained in that repository. You can follow the instructions provided by the repository supplier to retrieve the resource, if required.

   If you are using the local file system or a shared file system to store your resources, no copy of the resource is retained. Be very careful to select the correct resource when you complete this task.

Version and keyword information for deployable objects

Use the Broker Archive file editor to view the version and keyword information of deployable objects.

You can display properties of deployed objects, and can modify associated comments:

- "Displaying object version in the Broker Archive editor"
- "Displaying version, deploy time, and keywords of deployed objects" on page 266
- "Populating the Comment and Path columns" on page 266

Displaying object version in the Broker Archive editor

The column in the Broker Archive editor called Version displays the version tag for all objects that have a defined version:

- `.dictionary` files
- `.cmf` files
- Embedded JAR files with a version defined in a `META-INF/keywords.txt` file

You cannot edit the Version column.

You can use the mqsireadbar command to list the keywords that are defined for each deployable file within a deployable archive file.

Displaying version, deploy time, and keywords of deployed objects

The Properties View displays the following properties for all deployed objects:

- Version
- Deploy Time
- All defined keywords

For example, you deploy a message flow with the following literal strings:

- `$MQSI_VERSION=v1.0 MQSI$`
- `$MQSI Author=fred MQSI$`
- `$MQSI Subflow 1 Version=v1.3.2 MQSI$`

The Properties View displays these properties:

Deployment Time	Date and time of deployment
Modification Time	Date and time of modification
Version	`v1.0`
Author	`fred`
Subflow 1 Version	`v1.3.2`

If the keyword information is not available, a message is displayed in the Properties View to indicate the reason. For example, if keyword resolution has not been enabled at deploy time, the Properties View displays the message `Deployed with keyword search disabled`.

If you deploy to a Configuration Manager that is an earlier version than Version 6.0, the message is `Keywords not available on this Configuration Manager`.

Populating the Comment and Path columns

If you add source files, the Path column is populated automatically.

To add a comment, double-click the Comment column and type the text that you require.

Saving a message flow

You might want to save your message flow when you close the workbench, work with another resource, or validate the contents of the message flow.

Before you start:

To complete this task, you must have completed the following task:

- "Creating a message flow" on page 259

To save a message flow:

1. Switch to the Broker Application Development perspective.
2. Select the editor view that contains the open message flow that you want to save.

3. If you want to save the message flow without closing it in the editor view, press Ctrl+S or click **File** → **Save name** on the taskbar menu (where *name* is the name of this message flow). You can also save everything by clicking **File** → **Save All**.

   The message flow is saved and the message flow validator is invoked to validate its contents. The validator provides a report of all errors that it finds in the Problems view. The message flow remains open in the editor view.

   For example, if you save a message flow and have not set a mandatory property, an error message appears in the Problems view. The editor marks the node with the error icon  . The message flow in the Broker Development view is also marked with the error icon. This error can occur if you have not edited the properties of an MQInput node to define the queue from which the input node retrieves its input messages.

   (If you edit the properties of a node, you cannot click **OK** unless you have set all mandatory properties. Therefore this situation can arise only if you have never set properties.)

   You might also get warnings when you save a message flow. These are indicated by the warning icon  . This icon indicates that, although the configuration of the message flow does nto contain an explicit error, the configuration might result in unexpected results when the message flow completes. For example, if you have included an input node in your message flow that you have not connected to another node, you get a warning. In this situation, the editor marks the node with the warning icon. The message flow in the Broker Development view is also marked with a warning icon.
4. If you save a message flow that includes a subflow, and the subflow is no longer available, three error messages are added to the Problems view. These errors indicate that the input and output terminals and the subflow itself cannot be located. This error can occur if the subflow has been moved or renamed.

   To resolve this situation, right-click the subflow node in error and click **Locate Subflow**. The Locate Subflow dialog is displayed, listing the available message flow projects. Expand the list and explore the resources available to locate the required subflow. Select the correct subflow and click **OK**. All references in the current message flow are updated for you and the errors removed from the Problems view.
5. If you want to save the message flow when you close it, click the close view icon  on the editor view tab for this message flow or click **File** → **Close** on the taskbar menu. The editor view is closed and the file saved. The same validation occurs and all errors and warnings are written to the Problems view.

For information about using the **File** → **Save As** option to take a copy of the current message flow, see "Copying a message flow by using save."

See "Correcting errors from saving a message flow" on page 268 for information about handling errors from the save action.

Copying a message flow by using save

You can copy a message flow by using the **File** → **Save As** option.

1. Click **File** → **Save name As**.

2. Specify the message flow project in which you want to save a copy of the message flow. The project name defaults to the current project. You can accept this name, or choose another name from the valid options that are displayed in the File Save dialog.
3. Specify the name for the new copy of the message flow. If you want to save this message flow in the same project, you must either give it another name, or confirm that you want to overwrite the current copy (that is, copy the flow to itself).

   If you want to save this message flow in another project, the project must already exist (you can only select from the list of existing projects). You can save the flow with the same or another name in another project.
4. Click **OK**. The message flow is saved and the message flow editor validates its contents. The editor provides a report of all errors that it finds in the Problems view. See "Correcting errors from saving a message flow" for information about handling errors from the save action.

Correcting errors from saving a message flow

Correct the errors that are reported when you save a message flow.

To correct errors from the save or save as action:

1. Examine the list of errors and warnings that the validator has generated, which are displayed in the Problems view.
2. Double-click each entry in turn. The message flow is displayed in the editor view (if it is not already open in that view), and the editor selects the node in which the error was detected. If the error has been generated because you have not set a mandatory property, the editor also opens the Properties view, or dialog box, for that node.

   If you have included a user-defined node in your message flow, and have defined one or more of its properties as configurable, you might get a warning about a custom property editor. If you define a property as a configurable property, and you have specified that it uses a custom property editor, the Broker Archive editor cannot handle the custom property editor and handles the property as if it is type String. This action restricts your ability change this property at deployment time.
3. Correct the error that is indicated by the message. For example, provide a value for the mandatory property.
4. When you have corrected all the errors, save the message flow again. The editor validates all the resources that you have changed, and removes the corresponding graphical indication from the nodes that you have modified successfully. It also removes all corrected errors from the Problems view.

You do not have to correct every error to save your work. The editor saves your resources even if it detects errors or warnings, so that you can continue to work with them at a later date. However, you cannot deploy a resource that has a validation error. You must correct every error before you deploy a resource. Warnings do not prevent successful deployment.

Defining message flow content

You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

When you create a message flow, the editor view is initially empty. You must create the contents of the message flow by using a combination of the following tasks:

- "Adding a message flow node" on page 272
- "Adding a subflow" on page 275
- "Renaming a message flow node" on page 275
- "Configuring a message flow node" on page 276
- "Connecting message flow nodes" on page 280
- "Adding a bend point" on page 283
- "Aligning and arranging nodes" on page 285

When you finalize the contents of the message flow, you might also need to complete the following tasks:

- "Removing a message flow node" on page 279
- "Removing a node connection" on page 283
- "Removing a bend point" on page 284

To learn more about message flow content, you can import either of the following samples:

- Airline Reservations
- Error Handler

Follow the supplied instructions to build the sample yourself. You can also try adding and deleting nodes, adding subflows, and connecting nodes together.

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

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

Using the node palette

The node palette contains all the built-in nodes, which are organized into categories.

Before you start:

Read the following concept topic "Message flow node palette" on page 8.

You can add the nodes that you use most often to the Favorites category by following the instructions in "Adding nodes to the Favorites category on the palette" on page 271.

You can change the palette preferences in the Message Broker Toolkit. The changes that you can make are described in the following topics.

- "Changing the palette layout"
- "Changing the palette settings" on page 270
- "Customizing the palette" on page 270

Changing the palette layout

You can change the layout of the palette in the Message Flow editor and the Broker Topology editor.

1. Switch to the Broker Application Development perspective
2. Right-click the palette to display the pop-up menu.
3. Click **Layout**.
4. Click one of the available views:

   Columns
   Displays named icons in one or more columns. Change the number of columns by clicking on the right edge of the palette and dragging.

   List Displays named icons in a single-column list. The list view is the default layout.

   Icons Only
   Displays a list of icons only.

   Details
   Displays descriptions of the icons.

Changing the palette settings

How to use the **Palette Settings** dialog box.

Change the palette settings in the Message Flow editor and the Broker Topology editor using the **Palette Settings** dialog box.

1. Switch to the Broker Application Development perspective.
2. Right-click the palette to display the pop-up menu.
3. Click **Settings**. The **Palette Settings** dialog box opens.
4. Use the dialog to change the appropriate setting:
   - Click **Change** to change the font on the palette.
   - Click **Restore Default** to restore the default palette settings.
   - In the **Layout** list, click the appropriate radio button to change the palette layout. (See “Changing the palette layout” on page 269 for more information.)
   - Select **User large icons** to toggle between large and small icons in the palette.
   - In the **Drawer options** list, click the appropriate radio button to change the way that drawers are handled in the palette. A drawer is a container for a list of icons, such as the **Favorites** drawer on the Message Flow editor's palette, or the **Entity** drawer on the Broker Topology editor's palette.

Customizing the palette

If you customize the message flow node palette, you can make it easier to find the nodes that you use most often.

This saves time and on-screen space. For example:

- Change the order of the drawers in the palette so that the ones that you use most often are at the top.
- Hide any drawers that you do not use, to save on-screen space.
- Pin open the drawers that contain the nodes that you use most often.
- Create your own drawers to hold user-defined nodes that you create.

Customize the palette for the Message Flow editor using the Customize Palette dialog box:

1. Switch to the Broker Application Development perspective.
2. Right-click the palette, then click **Customize**. The Customize Palette dialog box opens.

- To change the order of entries and drawers in the palette, click the appropriate item in the list to highlight it, then click **Move Down** or **Move Up**. You cannot move any category above the Favorites category.
- To hide an entry or drawer, click the appropriate item in the list to highlight it, then select the **Hide** check box.
- To create a new separator, click **New → Separator**.
- To create a new drawer:
  a. Click **New → Drawer**.
  b. Type a name and description for the drawer.
  c. If required, select the **Open drawer at start-up** check box.
  d. If required, select the **Pin drawer open at start-up** check box.

3. Click **OK** or **Apply** to save your changes.

You have customized the message flow node palette.

Adding nodes to the Favorites category on the palette

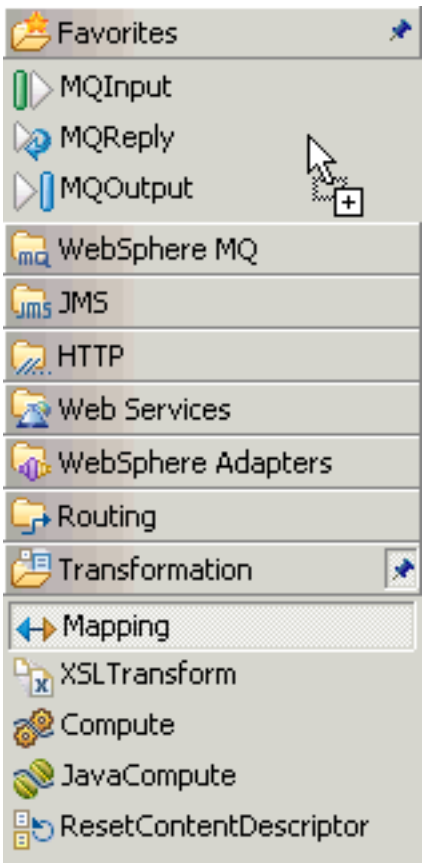
The nodes on the palette are organized in categories. The first category is Favorites, which is usually empty. You can drag the nodes that you use most often to the Favorites category.

Before you start:

Read the concept topic about the message flow node palette.

The following steps describe how to drag nodes into the Favorites category.

1. Switch to the Broker Application Development perspective.
2. On the palette, open the Favorites category.
3. On the palette, open the category that contains the node that you want to add to the Favorites category.
4. Use the mouse to drag the node into the Favorites category, as shown in the following example:



Alternatively, right-click the palette and choose the appropriate option to add or remove nodes from the Favorites category.

Adding a message flow node

When you have created a new message flow, add nodes to define its function.

Before you start:

- Create a message flow or open an existing message flow, see "Creating a message flow" on page 259 and "Opening an existing message flow" on page 261.
- Read the concept topic about message flow nodes, see "Message flow nodes" on page 6.

To add a node to a message flow:

1. Switch to the Broker Application Development perspective.
2. Open the message flow with which you want to work.
3. Open the Palette.
   - Hold the mouse pointer over the palette bar while it is in collapsed mode. The palette bar expands. When you move the mouse pointer away from the palette bar, it collapses again.
   - Click the **Show Palette** icon at the top of the palette bar. The palette bar expands and it remains expanded when the mouse is moved away from the palette bar. To collapse the palette bar again, click the **Hide Palette** icon at the top of the palette bar while it is in expanded mode.
4. Click **Selection** above the palette of nodes.

5. Decide which node you want to add: a built-in node or a user-defined node. You can select any of the nodes that are displayed in the node palette, but you can add only one node at a time.

   Nodes are grouped in categories according to the function that they provide. To see descriptions of the nodes in the palette, either hold the mouse pointer over a node in the palette, or switch to the Details view by following the instructions in "Changing the palette layout" on page 269.
6. Drag the node from the node palette onto the canvas.

   When you add a node to the canvas, the editor automatically assigns a name to the node, but the name is highlighted and you can change it by entering a name of your choice. If you do not change the default name at this time, you can changed it later, see "Renaming a message flow node" on page 275. The default name is set to the type of node for the first instance. For example, if you add an MQInput node to the canvas, it is given the name MQInput; if you add a second MQInput node, the default name is MQInput1; the third is MQInput2, and so on.
7. Repeat steps 5 and 6 to add further nodes.
8. You can also add nodes from other flows into this flow:
   a. Open the other message flow.
   b. Select the node or nodes that you want to copy from the editor or outline views, and press Ctrl+C or click **Edit** → **Copy**.
   c. Return to the flow with which you are currently working.
   d. Press Ctrl+V or click **Edit** → **Paste**. This action copies the node or nodes into your current flow. The node names and properties are preserved in the new copy.

When you have added the nodes that you want in this message flow, you can connect them to specify the flow of control through the message flow, and you can configure their properties.

Now you can configure the nodes, see "Configuring a message flow node" on page 276.

Adding a node by using the keyboard

You can use the keyboard to perform tasks in the Message Flow editor, such as adding a node to the canvas.

Before you start:

- Create a message flow or open an existing message flow.
- Read the concept topic about message flow nodes.

1. Switch to the Broker Application Development perspective.
2. Open the message flow to which you want to add a node.
3. Open the Palette view or the Palette bar.
4. Select a node in the Palette view or Palette bar by using the up and down arrows to highlight the node that you want to add to the canvas.
5. Add the node to the canvas by using one of the following methods:
   - Press **Alt + L**, then press **N**.
   - Press **Shift + F10** to open the pop-up menu for the Palette, and press **N**.

   The node that you selected in the Palette bar or Palette view is placed on the canvas in the Editor view.

When you add a node to the canvas, the editor automatically assigns a name to the node, but the name is highlighted and you can change it by entering a name of your choice. If you do not change the default name at this time, you can change it later by following the instructions in "Renaming a message flow node" on page 275. The default name is set to the type of node for the first instance. For example, if you add an MQInput node to the canvas, it is given the name MQInput; if you add a second MQInput node, the default name is MQInput1; the third is MQInput2, and so on.

You can move the node that you have placed on the canvas by using the keyboard controls described in Message Broker Toolkit keyboard shortcuts.

Dragging a resource from the Broker Development view

Drag a node or a related resource into the Message Flow editor.

Before you start:

- Create a message flow or open an existing message flow
- Read about message flow nodes

Drag a resource from the Broker Development view to an empty canvas to create a new node, or drag a resource onto an existing node to modify that node. The following resources are supported:

- An Adapter file
- An ESQL file
- A Java file
- A subflow
- A WSDL file
- An XSL file

1. Switch to the Broker Application Development perspective.
2. Open the message flow with which you want to work.
3. Drag one of the supported resources from the Broker Development view onto the canvas.
   - If you drop the resource on an empty canvas, a node is created and configured automatically.

     The following table shows the results when you drag a resource from the Broker Development view onto an empty canvas:

Resource	Node created	Property set
Adapter file	A "PeopleSoftInput node" on page 1115, "SAPInput node" on page 1147, or "SiebelInput node" on page 1165 is created	Adapter component
ESQL file	A "Compute node" on page 894 is created	ESQL Module
Java file	A "JavaCompute node" on page 1013 is created	Java Class
WSDL file	A "SOAPInput node" on page 1194 or "SOAPRequest node" on page 1204 is created	WSDL file name
XSL file	An "XSLTransform node" on page 1311 is created	Stylesheet

- If you drop the resource onto an existing node, the relevant node property is updated with the name of the resource file. For example, if you drop a Java file onto a JavaCompute node, the Java Class property is set to the class name of the Java file that you are dropping. If you drop an ESQL file over any node that uses ESQL, such as a Database node, the ESQL Module property is set.

Adding a subflow

In a message flow, you might want to include an embedded message flow, also known as a subflow. For example, you might define a subflow that provides error handling, and include it in a message flow connected to a failure terminal on a node that can generate an error in some situations.

Before you start

To complete this task, you must have completed one of the following tasks:
- "Creating a message flow" on page 259
- "Opening an existing message flow" on page 261

When you add a subflow, it is displayed in the editor view as a single node.

You can embed subflows into your message flow if either of the following statements is true:
- The flow that you want to embed is defined in the same message flow project.
- The flow is defined in a different message flow project, and you have specified the dependency of the current message flow project on that other project.

To add a subflow to a message flow:
1. Switch to the Broker Application Development perspective.
2. Open the message flow that you want to work with.
3. Drag the message flow from the Navigator view into the editor view. Alternatively, highlight the embedding message flow and click **Edit → Add subflow**, which displays a list of valid flows that you can add to the current flow.
4. Select the flow that you want to add from the list. The subflow icon is displayed with the terminals that represent the Input and Output nodes that you have included in the subflow.
5. Click **OK**.
6. Repeat steps 3, 4, and 5 to add further subflow nodes.
7. Select and open (double-click) the flow by name in the Navigator view, or right-click the embedded flow icon and select **Open Subflow** to work with the contents of the embedded flow

When you have added the nodes that you want in this message flow, you can connect them to specify the flow of control through the message flow. You can also modify their properties to suit your message processing requirements.

Renaming a message flow node

You can change the name of any type of node (a built-in node, user-defined node, or subflow node) to reflect its purpose.

Before you start:

- Create a message flow
- Read the concept topic about message flow nodes

When you first add a node to the canvas, the editor automatically assigns a name to the node, but the name is highlighted and you can change it by entering a name of your choice. If you do not change the default name at this time, you can change it later, as described in this topic. For example, you might include a Compute node to calculate the price of a specific part in an order, and you could change the name of the node to be Calculate_Price.

When you rename a node, use only the supported characters for this entity. The editor prevents you from entering unsupported characters.

To rename a node:

1. Switch to the Broker Application Development perspective.
2. Open the message flow with which you want to work.
3. You can rename a node in three ways:
   - Right-click the node and click **Rename**. The name is highlighted; enter a name of your choice and press Enter.
   - Click the node to select it, then click the node's name so that it is highlighted; enter a name of your choice and press Enter.
   - Click the node to select it, then on the Description tab of the Properties view, enter a name of your choice in the Node name field.

   The name that you enter must be unique in the message flow.

If you generate ESQL code for a Compute, Database, or Filter node, the code is contained in a module that is associated with the node. The name of the module in the ESQL file must match the name specified for the module in the *ESQL Module* property of the corresponding node. Although you can modify the module name, and change it from its default value (which is the name of the message flow, concatenated with the name of the node with which the module is associated), ensure that the module in the ESQL file matches the node property.

Configuring a message flow node

When you have included an instance of a node in your message flow, you can configure its properties to customize how it works.

Before you start:

- Read the concept topic about message flow nodes
- Add a node

Viewing a node's properties

To view a node's properties:

1. Switch to the Broker Application Development perspective.
2. Open the message flow with which you want to work.
3. Open the palette.
4. Click **Selection** above the node palette.
5. Right-click a node and click **Properties** to open the Properties view.

For nodes that do not have an associated resource, you can also double-click the node to display the properties. However, if you double-click any of the nodes in the following table, you open the associated resource.

Node	Result of double-clicking the node
"Compute node" on page 894	Opens an ESQL file
"Database node" on page 902	Opens an ESQL file
"DataDelete node" on page 923	Opens the New Message Map dialog box
"DataInsert node" on page 926	Opens the New Message Map dialog box
"DataUpdate node" on page 930	Opens the New Message Map dialog box
"Extract node" on page 944	Opens the New Message Map dialog box
"JavaCompute node" on page 1013	Opens the New JavaCompute Node Class wizard
"Mapping node" on page 1052	Opens the New Message Map dialog box
"SOAPAsyncRequest node" on page 1172	Opens the WSDL Selection dialog box
"SOAPInput node" on page 1194	Opens the WSDL Selection dialog box
"SOAPRequest node" on page 1204	Opens the WSDL Selection dialog box
"Warehouse node" on page 1307	Opens the New Message Map dialog box
WebSphere Adapters nodes	Opens the Adapter Component Selection dialog box
"XSLTransform node" on page 1311	Opens the XSL Selection dialog box

The selected node's properties are displayed.

Editing a node's properties

Properties are organized into related groups and displayed on tabs. Each tab is listed on the left of the Properties view. Click each tab to view the properties that you can edit.

- Every node has at least one tab, **Description**, where you can change the name of the node and enter short and long descriptions. The description fields are optional because they are used only for documentation purposes.
- If a property is mandatory, that is, one for which you must enter a value, the property name is marked with an asterisk, as shown in the following example:

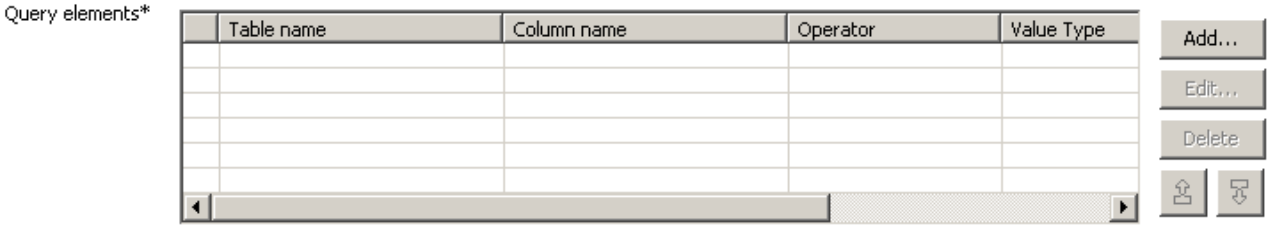
  Queue Name* _____

For details of how to configure each individual built-in node, see the node description. You can find a list of the nodes, with links to the individual topics, in "Built-in nodes" on page 875.

If you have included a user-defined node, refer to the documentation that came with the node to understand if, and how, you can configure its properties.

Editing complex properties

A complex property is a property to which you can assign multiple values. Complex properties are displayed in a table in the Properties view, where you can add, edit, and delete values, and change the order of the values in the table. This example shows the Query elements complex property of the DatabaseRoute node.



- To add a value to a complex property, click **Add**, enter the required fields in the dialog box that opens, then click **OK**. The values appear in the table. Repeat this step to enter as many values as are required.
- To edit a value, click any element in a row, click **Edit**, edit any of the values in the dialog box, then click **OK**.
- To delete a value, click any element in a row and click **Delete**. The entire row is deleted.
- To change the order of values in the table, click any element in a row and click the up icon  or down icon  to move the row.

Promoting properties

You can promote node properties to their containing message flow; for more information, see “Promoting a property” on page 643. Use this technique to set some values at the message flow level, without having to change individual nodes. This can be useful, for example, when you embed a message flow in another flow, and want to override some property such as output queue or data source with a value that is correct in this context. You cannot promote complex properties. For a full list of properties that are unavailable for promotion, as well as instructions for how to promote properties, see “Promoting a property” on page 643.

Overriding properties at deployment time

You can override a small number of node property values when you deploy a message flow. These property values are known as configurable properties, and you can use them to modify some characteristics of a deployed message flow without changing the message flow definitions. For example, you can update queue manager and data source information.

Even though you can set values for configurable properties at deployment time, you must set values for these properties within the message flow if they are mandatory. Each built-in node reference topic contains a table of properties, which identifies the configurable and mandatory properties.

Next: connect the nodes.

Using dynamic terminals

You can add, rename, and remove dynamic terminals on a node in the Message Flow editor.

Before you start:

- Add a node that supports dynamic terminals; for more details, see "Adding a message flow node" on page 272 and "Message flow node terminals" on page 68.

Some message flow nodes support dynamic input or output terminals, including the Collector, Route, and DatabaseRoute nodes. When you have added a node to the flow editor, you can add, remove, or change dynamic terminals.

- **Adding a dynamic terminal**
  1. Right-click the node and click **Add Input Terminal** or **Add Output Terminal**.
  2. Enter a name for the new terminal and click **OK**. The name must be unique for the terminal type. For example, if an input terminal called *In* already exists, you cannot create a dynamic input terminal with the name *In*.

     The new terminal is displayed on the node. If a node has five or more terminals, they are displayed as a terminal group. The following example shows a Route node with more than four output terminals.  To connect a particular output terminal, click the terminal group to open the Terminal Selection dialog box, or right-click the node and select **Create Connection**.
- **Renaming a dynamic terminal**
  1. Right-click the node and click **Rename Input Terminal** or **Rename Output Terminal**. These options are available only if you have added one or more appropriate terminals to this node.
  2. Select from the list the name of the terminal that you want to change. Only dynamic terminals are listed because you cannot change the name of a static terminal.
  3. Enter a new name for the terminal and click **OK**. Do not rename a dynamic terminal if one of the node properties is configured to use that name.
- **Removing a dynamic terminal**
  1. Right-click the node and click **Remove Input Terminal** or **Remove Output Terminal**, These options are available only if you have added one or more appropriate terminals to this node.
  2. Select from the list the name of the terminal that you want to remove and click **OK**. Only dynamic terminals are listed because you cannot remove a static terminal. Do not remove a dynamic terminal if one of the node properties is configured to use that terminal.

When you have added dynamic terminals to a node, connect them to other nodes in the message flow; for more information, see "Connecting message flow nodes" on page 280.

Removing a message flow node

When you have created and populated a message flow, you might need to remove a node to change the function of the flow, or to replace it with another more appropriate node. The node can be a built-in node, a user-defined node, or a subflow node.

Before you start:

- Add a node
- Add a subflow
- Read the concept topic about message flow nodes

To remove a node:

1. Switch to the Broker Application Development perspective.
2. Open the message flow that you want to work with.
3. Select the node in the editor view and press the Delete key.
4. Highlight the node and click **Edit** → **Delete**

   You can also right-click the node in the editor view and click **Delete**, or right-click the node in the Outline view and click **Delete**. The editor removes the node. If you have created any connections between that node and any other node, those connections are also deleted when you delete the node.
5. If you delete a node in error, you can restore it by right-clicking in the editor view and clicking **Undo Delete**. The node and its connections, if any, are restored.
6.

   You can also click **Edit** → **Undo Delete** or press Ctrl+Z.
7. If you undo the delete, but decide it is the correct delete action, you can right-click in the editor view and click **Redo Delete**.

   You can also click **Edit** → **Redo Delete**.

Connecting message flow nodes

When you include more than one node in your message flow, you must connect the nodes to indicate how the flow of control passes from input to output. The nodes can be built-in nodes, user-defined nodes, or subflow nodes.

Before you start:

- Add a node
- Add a subflow
- Read the concept topic about connections

Your message flow might contain just one MQInput node, one Compute node, and one MQOutput node. Or it might involve a large number of nodes, and perhaps embedded message flows, that provide a number of paths through which a message can travel depending on its content. You might also have some error processing routines included in the flow. You might also need to control the order of processing.

You can connect a single output terminal of one node to the input terminal of more than one node (this is known as fan-out). If you do this, the same message is propagated to all target nodes, but you have no control over the order in which the subsequent paths through the message flow are executed (except with the FlowOrder node).

You can also connect the output terminal of several nodes to a single node input terminal (this is known as fan-in). Again, the messages that are received by the target node are not received in any guaranteed order.

When you have completed a connection, it is displayed as a black line, and is drawn as close as possible to a straight line between the connected terminals. This behavior might result in the connection passing across other nodes. To avoid this problem, you can add bend points to the connection.

In the Message Flow editor, you can display node and connection metadata by holding the mouse pointer over a node or subflow in a message flow. To view metadata information for a node, subflow, or connection:

1. Switch to the Broker Application Development perspective.
2. Open a message flow.
3. In the Message Flow editor, hold the mouse pointer over a node, a subflow, or a node connection in the open message flow by placing the mouse over the element.

A custom tooltip is displayed below the element.

- To turn the pop-up window into a scrollable window, press **F2**.
- To hide the pop-up window, either press Esc or move the mouse away from the node.

If you define a complex message flow, you might have to create a large number of connections. The principle is the same for every connection. You create connections either by using the mouse, or by using the Terminal Selection dialog. See "Creating node connections with the mouse" and "Creating node connections with the Terminal Selection dialog box" on page 282 for more information.

Creating node connections with the mouse

Use the mouse to connect one node to another.

Before you start:

Read the concept topic about connections.

1. Switch to the Broker Application Development perspective.
2. Open the message flow with which you want to work.
3. Click the terminal from which the connection is to be made; that is, the terminal from which the message is propagated from the current node.

   For example, you can click the Failure, Out, or Catch terminal of the MQInput node. Hold the mouse pointer over each terminal to see the name of the terminal. You do not need to keep the mouse button pressed.

   Alternatively, click **Connection** on the palette, then click the node from which the connection is to be made. The Terminal Selection dialog box opens for you to choose the terminal from which to make a connection. Click **OK**. If a node has five or more input or output terminals (for example, if you have added dynamic terminals), they are displayed in a group. The following example shows a node with more than four output nodes.  To select a particular output terminal, click the grouped output terminal to open the Terminal Selection dialog box.
4. Click the input terminal of the next node in the message flow (to which the message passes for further processing). The connection is made when you click a valid input terminal. The connection appears as a black line between the two terminals.

In the Message Flow editor, you can display node and connection metadata by holding the mouse pointer over a node or subflow in a message flow. To view metadata information for a node, subflow, or connection:

1. Switch to the Broker Application Development perspective.
2. Open a message flow.
3. In the Message Flow editor, hold the mouse pointer over a node, a subflow, or a node connection in the open message flow by placing the mouse over the element.

A custom tooltip is displayed below the element.

- To turn the pop-up window into a scrollable window, press **F2**.
- To hide the pop-up window, either press Esc or move the mouse away from the node.

Next: add a bend point, as described in "Adding a bend point" on page 283.

Creating node connections with the Terminal Selection dialog box

Use the Terminal Selection dialog box to connect one node to another.

Before you start:

Read the concept topic about connections.

1. Switch to the Broker Application Development perspective.
2. Open the message flow with which you want to work.
3. Click **Connection** above the node palette.
4. Click the node from which you want the connection to be made. The Terminal Selection dialog box is displayed.
5. Select the terminal from the list of valid terminals on this node. Click **OK**. The dialog box closes.
6. Click the node to which to make the connection. If this node has only one input terminal, the connection is made immediately. If this node has more than one input terminal, the Terminal Selection dialog box is displayed again, listing the input terminals of the selected node. Click the correct terminal and click **OK**.

Alternatively, you can make a connection in the following way:

1. Click **Selection** above the node palette.
2. Right-click the node from which you want to make the connection and click **Create Connection**. The Terminal Selection dialog box is displayed.
3. Select the terminal from the list of valid terminals on this node. Click **OK**. The dialog box closes.
4. Click the node to which to make the connection. If this node has only one input terminal, the connection is made immediately. If this node has more than one input terminal, the Terminal Selection dialog box is displayed again, listing the input terminals of the selected node. Click the correct terminal and click **OK**.

In the Message Flow editor, you can display node and connection metadata by holding the mouse pointer over a node or subflow in a message flow. To view metadata information for a node, subflow, or connection:

1. Switch to the Broker Application Development perspective.

2. Open a message flow.
3. In the Message Flow editor, hold the mouse pointer over a node, a subflow, or a node connection in the open message flow by placing the mouse over the element.

A custom tooltip is displayed below the element.

- To turn the pop-up window into a scrollable window, press **F2**.
- To hide the pop-up window, either press Esc or move the mouse away from the node.

Next: add a bend point, as described in "Adding a bend point."

Removing a node connection

The message flow editor displays the nodes and connections in the editor view. You can remove connections to change the way in which the message flow processes messages.

Before you start:
- Connect the nodes
- Read the concept topic about connections

To remove a connection that you have created between two nodes:

1. Switch to the Broker Application Development perspective.
2. Open the message flow that you want to work with.
3. Click **Selection** above the node palette.
4. Select the connection that you want to delete. When you hold the mouse pointer over the connection, the editor highlights the connection that you have selected by thickening its line, adding an arrowhead at the target terminal end, and annotating the connection with the name of the two terminals connected, for example Out->In.

   When you select the connection, the editor appends a small black square at each end and at every bend point of the connection, and a small arrowhead at the target terminal end. The annotation disappears when you select the connection.
5. Check that the selected connection is the one that you want to delete.
6. Right-click the connection and click **Delete**, press the Delete key, or click **Edit → Delete**. If you want to delete further connections, repeat these actions from step 4.
7. If you delete a connection in error, you can restore it by right-clicking in the editor view and clicking **Undo Delete**. The connection is restored.
8. If you undo the delete, but decide that it is the correct delete action, you can right-click in the editor view and click **Redo Delete**. You can also delete a connection by selecting it in the Outline view and pressing the Delete key.

If you delete a node, its connections are automatically removed; you do not have to do this as a separate task.

Adding a bend point

When you are working with a message flow, and connecting your chosen nodes together to determine the flow of control, you might find that a connection that you have made crosses over an intervening node and makes the flow of control difficult to follow. To help you to display the message flow nodes and their

connections in a clear way, you can add bend points to the connections that you have made to improve the organization of the display. The addition of bend points has no effect on the execution of the nodes or the operation of the message flow.

Before you start:
- Connect the nodes
- Read the concept topic about bend points

To add a bend point:

1. Switch to the Broker Application Development perspective.
2. Open the message flow that you want to work with.
3. Click **Selection** above the node palette.
4. Select the connection to which you want to add a bend point. The editor appends a small black square to each end of the connection to highlight it.
   a. Check that this is the correct connection. The editor also adds a small point (a handle) in the connection halfway between the in and out terminals that are joined by this connection.
5. Hold the mouse pointer over this point until the editor displays a black cross to indicate that you now have control of this bend point.
   a. Hold down the left mouse button and move your mouse to move the black cross and bend point across the editor view.
6. As you drag your mouse, the connection is updated, retaining its start and end points with a bend point at the drag point. You can move the bend point anywhere in the editor view to improve the layout of your message flow.
7. Release the mouse button when the connection is in the correct place. The editor now displays the bend point that you have created with a small square (such as those at the ends of the connection), and displays another two small points in the connection, one between your newly-created bend point and the out terminal, the other between the new bend point and the in terminal.

If you want to add more than one bend point to the same connection, repeat these actions from step 4 using the additional small points inserted into the connection.

Next: align and arrange the nodes.

Removing a bend point

When you are working with a message flow in the editor view, you might want to simplify the display of the message flow by removing a bend point that you previously added to a connection between two nodes.

Before you start:
- Add a bend point
- Read the concept topic about bend points

To remove a bend point:

1. Switch to the Broker Application Development perspective.
2. Open the message flow that you want to work with.
3. Click **Selection** above the node palette.
4. Select the connection from which you want to remove the bend point. The editor highlights the connection and its current bend points by thickening its

line and appending a small black square to each end of the connection, and by indicating each bend point with a small black square. Check that this is the correct connection.

5. Right-click over the selected connection, if you added this bend point in the current edit session.
   a. Click **Undo Create Bend Point**.

   The editor removes the selected bend point.

   If you right-click in the editor view without a connection being selected, you can also click **Undo Create Bend Point** from the menu. However, this removes the last bend point that you created in any connection, which might not be the one that you want to remove.
6. Move the bend point to straighten the line if you added this bend point in a previous edit session, because you cannot use the undo action. When the line is straight, the bend point is removed automatically.

   When the bend point has been removed, the connection remains highlighted. Both ends of the connection, and any remaining bend points, remain displayed as small black squares. The editor also inserts small points (handles) into the connection between each bend point and between each terminal and its adjacent bend point, which you can use to add more bend points.
7. If you want to remove another bend point from the same connection, repeat these actions from step 4 on page 284.

Aligning and arranging nodes

When you are working in the Message Flow editor, you can decide how your nodes are aligned in the editor view.

This option is closely linked to the way in which your nodes are arranged. The default for both alignment and arrangement is left to right, which means that the in terminal of a node appears on its left edge, and its out terminals appear on its right edge. You can also change this characteristic of a node by rotating the icon display to right to left, top to bottom, and bottom to top.

Before you start

To complete this task, you must have completed the following task:

- "Adding a message flow node" on page 272

To modify the way in which nodes and connections are displayed in the editor:

1. Switch to the Broker Application Development perspective.
2. Open the message flow that you want to work with.
3. Click **Selection** above the node palette.
4. Right-click in the editor window and select **Manhattan Layout** if you want the connections between the nodes to be displayed in Manhattan style; that is with horizontal and vertical lines joined at right angles.
5. If you want to change the layout of the complete message flow:
   a. Right-click in the editor view and click **Layout**. The default for the alignment is left to right, such that your message flow starts (with an input node) on the left and control passes to the right.
   b. From the four further options displayed, **Left to Right**, **Right to Left**, **Top to Bottom**, and **Bottom to Top**, click the option that you want for this

message flow. The message flow display is updated to reflect your choice. As a result of the change in alignment, all the nodes in the message flow are also realigned.

For example, if you have changed from a left to right display (the default) to a right to left display, each node in the flow has now also changed to right to left (that is, the in terminal now appears on the right edge, the out terminals appear on the left edge).

6. You might want to arrange an individual node in a different direction from that in which the remaining nodes are arranged in the message flow:
   a. Right-click the node that you want to change and click **Rotate**. Four further options are displayed: **Left to Right**, **Right to Left**, **Top to Bottom**, and **Bottom to Top**. The option that represents the current arrangement of the node is not available for selection.
   b. Click the option that you want for this node.

If you change the alignment of the message flow, or the arrangement of an individual node, or both, these settings are saved when you save the message flow. They are applied when another user accesses this same message flow, either through a shared repository or through shared files or import and export. When you reopen the message flow, you see these changed characteristics. The alignment and arrangement that you have selected for this message flow have no effect on the alignment and arrangement of any other message flow.

In the Message Broker Toolkit Version 5.1 you can adjust the zoom by right-clicking in the editor view and clicking **Zoom in** or **Zoom out**. Alternatively, you can use the drop-down list on the editor toolbar to specify a zoom percentage.

You can also access the editor toolbar to select other options related to the display and arrangement of nodes, for example, snap to grid. These options are defined in Message Flow editor.

Developing message flow applications that use WebSphere Adapters

For information about how to develop message flow applications that use WebSphere Adapters, see the following topics.

Preparatory tasks

- "Preparing your system to use WebSphere Adapters nodes" on page 287
- "Activating IBM Tivoli License Manager for WebSphere Adapters" on page 288

SAP

- "Adding external software dependencies for SAP" on page 288
- "Configuring the SAP server to work with the adapter" on page 290
- "Changing connection details for SAP adapters" on page 299

Siebel

- "Adding external software dependencies for Siebel" on page 291
- "Configuring the Siebel application to work with the adapter" on page 293
- "Changing connection details for Siebel adapters" on page 300

PeopleSoft

- "Adding external software dependencies for PeopleSoft" on page 295

- "Creating a custom event project in PeopleTools" on page 296
- "Changing connection details for PeopleSoft adapters" on page 301

All systems

- "Connecting to an EIS by using the Adapter Connection wizard" on page 298

Preparing your system to use WebSphere Adapters nodes

Before you can connect to an Enterprise Information System (EIS), you must prepare your system by adding external software dependencies and configuring the EIS to work with the WebSphere Adapter.

Before you start:

- For general background information, read "WebSphere Adapters nodes" on page 10
- Check for the latest information about WebSphere Adapters at WebSphere Adapters technotes.
- Check for the latest information about support for adapters on different operating systems at WebSphere Message Broker Requirements.
- Check the mode of your broker, because it can affect the number of execution groups and message flows that you can deploy, and the types of node that you can use. For more information, see Restrictions that apply in each operation mode and Checking the operation mode of your broker.
- Enable the WebSphere Adapters nodes in the broker runtime environment; see Preparing the environment for WebSphere Adapters nodes.
- If you want to use the IBM Tivoli® License Manager (ITLM), perform the steps in "Activating IBM Tivoli License Manager for WebSphere Adapters" on page 288.
- To see how the WebSphere Adapters work, look at the following samples:
  - Twineball Example EIS Adapter
  - SAP Connectivity

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

Perform the following steps, in the order shown, to prepare your system to use WebSphere Adapter nodes.

- **SAP**
  1. Follow the instructions in "Adding external software dependencies for SAP" on page 288.
  2. Follow the instructions in "Configuring the SAP server to work with the adapter" on page 290.
- **Siebel**
  1. Follow the instructions in "Adding external software dependencies for Siebel" on page 291.
  2. Follow the instructions in "Creating the event store manually" on page 293.
- **PeopleSoft**
  1. Follow the instructions in "Adding external software dependencies for PeopleSoft" on page 295.
  2. Follow the instructions in "Creating a custom event project in PeopleTools" on page 296.

After you have prepared your system, connect to an EIS by following the instructions in "Connecting to an EIS by using the Adapter Connection wizard" on page 298.

Activating IBM Tivoli License Manager for WebSphere Adapters

If you want to use IBM Tivoli License Manager (ITLM), you must activate it for each of the WebSphere Adapters.

ITLM enables you to monitor the usage of IBM (and other) software products. For more information, see the IBM Tivoli License Manager Information Center or the IBM Tivoli License Manager Web site.

The following steps describe how to activate the ITLM file for each of the adapters.

1. Locate the ITLM directory for the adapter.
   - For SAP: *install_dir*/itlm/SAP
   - For Siebel: *install_dir*/itlm/Siebel
   - For PeopleSoft: *install_dir*/itlm/PeopleSoft
2. Remove the inactive file extension from the file in the ITLM directory so that it ends with .sys2.

After you have performed these steps, when you run ITLM, the adapter is visible.

Adding external software dependencies for SAP

Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

Before you start:

Ensure that you have the relevant prerequisite files for your SAP system.

- sapjco.jar
- Windows On Windows:
  - sapjcorfc.dll
  - librfc32.dll
- z/OS Linux UNIX On z/OS, Linux, and UNIX:
  - libsapjcorfc.so
  - librfccm.so

Download these files for your operating system from the external SAP Web site, SAP Service Marketplace, and save them to a directory, such as C:\SAP_LIB. (On Windows, the directory cannot be a mapped network drive on a remote Windows computer; the directory must be local or on a Storage Area Network (SAN) disk.) You must have an SAPNet account to be able to access this Web site.

- Windows On Windows, download the .dll files that come with the SAP JCo download.
- z/OS Linux UNIX On z/OS, Linux, and UNIX, download the .so and .o files that come with the SAP JCo download.

Locating the SAP support files in the runtime environment

To add the SAP prerequisite files to the runtime environment, take the following steps.

- Windows Linux UNIX On Windows, Linux and UNIX:
  1. Ensure that the broker has started.
  2. Either open the Command Console, or open a Windows command prompt and enter mqsiprofile to initialize the environment.
  3. Enter the following command to display the locations of the prerequisite JAR files and native libraries:

     ```
     mqsireportproperties WBRK61_DEFAULT_BROKER -c EISProviders -o AllReportableEntityNames  -r
     ```

     The following example shows what typically is displayed when you run this command:

     ```
     ReportableEntityName=''
      EISProviders
       PeopleSoft=''
        jarsURL='default_Path'
        nativeLibs='default_Path'
       SAP=''
        jarsURL='default_Path'
        nativeLibs='default_Path'
       Siebel=''
        jarsURL='default_Path'
        nativeLibs='default_Path'
       Twineball=''
        jarsURL='default_Path'
        nativeLibs='default_Path'
     ```
  4. Set the location of the SAP prerequisite files using the following command:

```
mqsichangeproperties WBRK61_DEFAULT_BROKER -c EISProviders -o SAP  -n jarsURL -v C:\SAP_LIB
mqsichangeproperties WBRK61_DEFAULT_BROKER -c EISProviders -o SAP  -n nativeLibs -v C:\SAP_LIB
```

  5. To check that the values have been set correctly, run the following command:

     ```
     mqsireportproperties  WBRK61_DEFAULT_BROKER -c EISProviders -o SAP -r
     ```

     The following example shows what is displayed by the mqsireportproperties command.

     ```
     ReportableEntityName=' '
     EISProviders
      SAP=' '
       jarsURL='C:\SAP_LIB'
       nativeLibs='C:\SAP_LIB'

     BIP8071I: Successful command completion.
     ```
  6. Restart the broker.
- z/OS On z/OS: Run the mqsireportproperties command by customizing and submitting the BIPCHPR utility. For more information about this utility, see Contents of the broker PDSE.

Locating the SAP support files through the Message Broker Toolkit

To add the SAP prerequisite files through the Message Broker Toolkit, take the following steps.

- Windows On Windows: When you run the Adapter Connection wizard, you are prompted to specify the paths to the required libraries.
- Linux On Linux: Append the directory that contains the SAP libraries to the LD_LIBRARY_PATH environment variable:

  ```
  LD_LIBRARY_PATH=DIRECTORY_CONTAINING_SAP_LIBRARIES:${LD_LIBRARY_PATH}
  export LD_LIBRARY_PATH
  ```

You must set this variable either for the whole system, or in the same shell or environment from which the Message Broker Toolkit is launched.

Next: configure the SAP system to work with the adapter

Configuring the SAP server to work with the adapter

Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

Before you start:

Add the required external software dependencies for SAP.

Perform the following steps on the SAP server using the SAP graphical user interface.

1. Register an RFC program ID:
   a. Open transaction **SM59** (Display and Maintain RFC Destinations).
   b. Click **Create**.
   c. Type a name for the RFC destination.
   d. In the **Connection Type** field, select **T**.
   e. In the **Activation Type** field, select **Registered Server Program**.
   f. Type a Program ID.

      You use this program ID when you configure the adapter. This value indicates to the SAP gateway which RFC-enabled functions the program ID listens for.
   g. Enter a description in **Description 1**, such as `RFC for Test Sample`.
   h. Enter a description in **Description 2**, such as your name.
   i. Click **MDMP & Unicode**, and in the Communication Type with Target System section, click **Unicode**.
   j. Save your entry.
2. Set up a receiver port:
   a. Open transaction **WE21** (Ports in IDoc processing).
   b. Select **Transactional RFC**, click **Ports**, and click the **Create** icon.
   c. Type a name for the port and click **OK**.
   d. Type the name of the destination that you created in the previous task (or select it from the list).
   e. Save your entry.
3. Specify a logical system:
   a. Open transaction **BD54** (Change View Logical Systems).
   b. Click **New Entries**.
   c. Type a name for the logical system and click the **Save** icon.
   d. If you see the Prompts for Workbench request, click the **New Request** icon. Then enter a short description and click **Save**.
   e. Click the **Continue** icon.
4. Configure a distribution model:
   a. Open transaction **BD64** (Maintenance of Distribution Model).

b. Click **Distribution Model → Switch processing model**.
c. Click **Create model view**.
d. Type a name for the model view and click the **Continue** icon.
e. Select the distribution model that you created and click **Add message type**.
f. For outbound processing, type the logical system name that you created in the previous task as **Sender**, and type the logical name of the SAP server as **Receiver**, then select a message type (for example, **MATMAS**) and click the **Continue** icon.
g. Select the distribution model again and click **Add message type**.
h. For inbound processing, type the logical name of the SAP server as **Sender**, and the logical system name that you created in the previous task as **Receiver**, then select a message type (for example, **MATMAS**) and click the **Continue** icon.
i. Save your entry.

5. Set up a partner profile:
   a. Open transaction **WE20** (Partner Profiles).
   b. Click the **Create** icon.
   c. Type the name of the logical system that you created in the earlier task and, for **Partner Type**, select **LS**.
   d. For **Post Processing: permitted agent**, type `US` and your user ID.
   e. Click the **Save** icon.
   f. In the Outbound parameters section, click the **Create outbound parameter** icon.
   g. In the Outbound parameters window, type a message type (for example, `MATMAS05`), select the receiver port that you created in the earlier task, and select **Transfer IDoc immed**.
   h. Click the **Save** icon.
   i. Press **F3** to return to the Partner Profiles view.
   j. In the Inbound parameters section, click the **Create inbound parameter** icon.
   k. In the Inbound parameters window, type a message type (for example, `MATMAS`), and a process code (for example, `MATM`).
   l. Click the **Save** icon.
   m. Press **F3** to return to the Partner Profiles view.
   n. In the Inbound parameters section, click the **Create inbound parameter** icon.
   o. In the Inbound parameters window, type the following values: `ALEAUD` for **Message Type**, and `AUD1` for **Process Code**.
   p. Click the **Save** icon.
   q. Press **F3** to return to the Partner Profiles view.
   r. Click the **Save** icon.

Next: connect to an EIS using the Adapter Connection wizard.

Adding external software dependencies for Siebel

Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

Before you start:

Ensure that you have the relevant prerequisite files for your Siebel system.

- Siebel Business Applications Versions 7.5 and earlier
  - SiebelJI_*language code*.jar (for example, SiebelJI_enu.jar)
  - SiebelJI_Common.jar
- Siebel Business Applications Versions 7.7x, 7.8x, and 8.0
  - Siebel.jar
  - SiebelJI_*language code*.jar (for example, SiebelJI_enu.jar)

Download these files from the Siebel application, and save them to a directory, such as C:\Siebel_LIB. (On Windows, the directory cannot be a mapped network drive on a remote Windows computer; the directory must be local or on a Storage Area Network (SAN) disk.)

The sample resources that you need to set up the Siebel system so that it can communicate with the broker are in the following directory: *install_dir*\ ResrouceAdapters\Siebel_6.1.0.0\samples.

Locating the Siebel support files in the runtime environment on Windows

To add the Siebel prerequisite files to the runtime environment, take the following steps.

1. Ensure that the broker has started.
2. Either open the Command Console, or open a Windows command prompt and enter mqsiprofile to initialize the environment.
3. Enter the following command to display the locations of the prerequisite JAR files and native libraries:

   ```
   mqsireportproperties WBRK61_DEFAULT_BROKER -c AllTypes -o AllReportableEntityNames  -r
   ```

   The following example shows what typically is displayed when you run this command:

   ```
   ReportableEntityName=''
    EISProviders
     PeopleSoft=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
     SAP=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
     Siebel=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
     Twineball=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
   ```

4. Set the location of the Siebel prerequisite files using the following command:

```
mqsichangeproperties WBRK61_DEFAULT_BROKER -c EISProviders -o Siebel -n jarsURL -v C:\Siebel_LIB
mqsichangeproperties WBRK61_DEFAULT_BROKER -c EISProviders -o Siebel -n nativeLibs -v C:\Siebel_LIB
```

5. To check that the values have been set correctly, run the following command:

   ```
   mqsireportproperties WBRK61_DEFAULT_BROKER -c EISProviders -o Siebel -r
   ```

   The following example shows what is displayed by the mqsireportproperties command.

   ```
   ReportableEntityName=' '
   EISProviders
    Siebel=' '
   ```

```
  jarsURL='C:\Siebel_LIB'
  nativeLibs='C:\Siebel_LIB'

BIP8071I: Successful command completion.
```

6. Restart the broker.

Next: configure the Siebel application to work with the adapter.

Configuring the Siebel application to work with the adapter

To configure the Siebel application, create an event table and a Siebel business object.

Before you start:

1. Add the required external software dependencies for Siebel.
2. Before you configure the Siebel application to work with WebSphere Adapter for Siebel Business Applications, you must create a user name and password so that the Adapter Connection wizard can connect to Siebel Business Applications to perform outbound operations, and retrieve Siebel business objects and services.

   You perform this task on the Siebel server, therefore ensure that you are familiar with the Siebel tools that are required to complete it. For information about using Siebel tools, refer to the Siebel tools documentation.

   To open Siebel Sales Enterprise on your local database, you must have administrative privileges.

To configure the Siebel application, you must create an event table and a Siebel business object. WebSphere Message Broker contains resources that help you to create the event components and triggers. This topic describes how to use those resources. You can also create the event table and Siebel business object manually; for more information, see "Creating the event store manually."

1. Locate the samples folder at *install_dir*/WMBT610/ResourceAdapters/Siebel/samples.

   The samples folder contains two folders: Siebel7.x.x and Siebel8.0. Each version has an Event_pkg folder, which contains a `.sif` file and a number of `.js` scripts. You use the `.sif` file to create the event components; it can add business objects, views, and all other Siebel objects to the Siebel repository. The `.js` scripts help you to create Siebel triggers.
2. To use the `.sif` file:
   a. Open Siebel tools and click **Tools** → **Import**.
   b. Import the `.sif` file.
   c. Merge the differences between the `.sif` file and the Siebel repository.
   d. Recompile the repository into a Siebel repository file (`.srf` file).
3. Use the `.js` scripts to create Siebel triggers. The provided samples show how to create entries in the inbound table when new Account objects are created.

Creating the event store manually

To configure the Siebel application, create an event table and a Siebel business object.

"Configuring the Siebel application to work with the adapter" describes how to use the samples that are supplied with WebSphere Message Broker to configure the Siebel application. This topic describes how to create the event store manually.

The following steps describe how to create the event store to be used for inbound operations in the Siebel application. You can substitute all references to Siebel Sales Enterprise with the name of the Siebel application that you are using.

1. Create a project called IBM, and lock the project with Siebel tools.
2. Using the object wizard, create an event table called CX_IBM_EVENT in which to store the events.
   a. In the event table, create the columns that are shown in the following table.

Column Name	Type	Length	Data Type	Required	Nullable	Status
DESCRIPTION	Data (public)	255	Varchar	No	Yes	Active
EVENT_ID	Data (public)	30	Varchar	Yes	No	Active
EVENT_TYPE	Data (public)	20	Varchar	Yes	No	Active
OBJECT_KEY	Data (public)	255	Varchar	Yes	No	Active
OBJECT_NAME	Data (public)	255	Varchar	Yes	No	Active
PRIORITY	Data (public)	10	Varchar	No	Yes	Active
STATUS	Data (public)	20	Varchar	Yes	No	Active
XID	Data (public)	255	Varchar	Yes	No	Active

   b. Create a new business component called IBM Event.
   c. Create a new time stamp called Field Event, and map it to the CREATED column from CX_IBM_EVENT. Make the Type of this field DTYPE_UTCDATETIME.
   d. Create a new business object called IBM Event.
   e. Associate the IBM event business component to the IBM Event business object.
   f. Create an applet called IBM Event List Applet, and base it on the IBM Event business component that you have created.
   g. Create a view called IBM Event List View, and base it on the IBM Event business object that you have created.
   h. Create a screen called IBM Event Screen, and associate it with the IBM Event List View in the Siebel tools.
3. Create a page tab.
   a. Click **Start Application → Siebel Sales Enterprise**.
   b. Right-click the **Page** tab, and click **New Record**.
   c. Specify IBM Event as the screen name, and IBM Event for the **Text - String Override** field.
   d. Leave the **Inactive** field blank.
4. Create a new business object called Schema Version for your IBM project and associate it with the Schema Version business component.
   a. Apply the physical schema for the new tables to your local database by querying for the new table, CX_IBM_EVENT_Q and selecting the current query to create a physical schema. Leave the table space and index space blank.
   b. Click **Activate** to activate the new schema.
5. Add or modify the Siebel VB or e-scripts for the business component that corresponds to the business objects that are used at your site. Siebel scripts trigger event notification for business objects. Samples are located in the Samples folder in your adapter installation.

6. Create a new Siebel repository file by compiling the updated and locked projects on your local database. The new repository file has an extension of `.srf`.
7. Create and populate a new responsibility.
   a. Open Siebel Sales Enterprise on your local database.
   b. Create a new responsibility called IBM Responsibility for IBM Event List View.
   c. Add the employees or teams who are responsible for reviewing events to the newly created IBM Responsibility.
   d. Create a user name called IBMCONN (or another user name to be used by the adapter later). Add the user name to the newly created IBM Responsibility and also to the Administrative Responsibility.
8. Test the application in your local environment to ensure that you can see the IBM Event List View. An event is generated in the view after you create a record in the supported object. As part of the test, create a new Account business component instance in Siebel. Confirm that a new Account event is shown in the IBM Event List View (assuming that you have added the e-script trigger to the Account business component). If a new Account event is not displayed in the view, check for an error and fix it. For more information on the errors that might be generated, check either the Siebel support site or Siebel documentation.
9. When the test that you perform in Step 8 is successful, add your new and updated projects to your development server.
10. Activate the new table in the development server.
11. Compile a new Siebel repository (`.srf`) file on the server.
12. Back up the original repository file on the server.
13. Stop the Siebel server and replace the original repository file with the newly created one.
14. Restart the Siebel server.

Adding external software dependencies for PeopleSoft

Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

Before you start:

Ensure that you have the relevant prerequisite files for your PeopleSoft system.

- `psjoa.jar`
- A JAR file that contains the component interface API classes

Save both support files to a directory such as `C:\PeopleSoft_LIB.` (On Windows, the directory cannot be a mapped network drive on a remote Windows computer; the directory must be local or on a Storage Area Network (SAN) disk.) You can find the `psjoa.jar` file in the following location on the PeopleSoft Application Server: *peopleTools_installation_directory*`\web\PSJOA\psjoa.jar`. Use PeopleTools to generate the component interface JAR file for your business objects.

The sample resources that you need to set up the PeopleSoft system so that it can communicate with the broker are in *install_dir*`\ResrouceAdapters\PeopleSoft_6.1.0.0\samples`.

Locating the PeopleSoft support files in the run time on Windows

To add the PeopleSoft prerequisite files to the run time, complete the following steps.

1. Ensure that the broker has started.
2. Either open the Command Console, or open a Windows command prompt and enter mqsiprofile to initialize the environment.
3. Enter the following command to display the locations of the prerequisite JAR files and native libraries:

   ```
   mqsireportproperties WBRK61_DEFAULT_BROKER -c AllTypes -o AllReportableEntityNames  -r
   ```

   The following example shows what typically is displayed when you run this command:

   ```
   ReportableEntityName=''
    EISProviders
     PeopleSoft=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
     SAP=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
     Siebel=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
     Twineball=''
      jarsURL='default_Path'
      nativeLibs='default_Path'
   ```

4. Set the location of the PeopleSoft prerequisite files by using the following command:

```
mqsichangeproperties WBRK61_DEFAULT_BROKER -c EISProviders -o PeopleSoft -n jarsURL -v C:\PeopleSoft_LIB
```

5. To check that the values have been set correctly, run the following command:

   ```
   mqsireportproperties WBRK61_DEFAULT_BROKER -c EISProviders -o PeopleSoft -r
   ```

   The following example shows what is displayed by the mqsireportproperties command.

   ```
   ReportableEntityName=' '
   EISProviders
    PeopleSoft=' '
     jarsURL='C:\PeopleSoft_LIB'

   BIP8071I: Successful command completion.
   ```

6. Restart the broker.

Next: create a custom event project in PeopleTools.

Creating a custom event project in PeopleTools

The WebSphere Adapter requires an event project in PeopleSoft to perform asynchronous inbound event processing. Use PeopleTools to create the custom event project.

Before you start:

Add the required external software dependencies for PeopleSoft.

If your environment requires inbound event support, you must use a custom event project in PeopleSoft. A sample event project, IBM_EVENT_V600, is provided with

the adapter. You can modify and use and the sample project, or you can create your own project by using PeopleTools. If you create your own project, make sure that you complete the following steps.

1. Use PeopleTools Application Designer to create and name a new project.
2. Create the fields for the new project as described in the following table:

Field name	Field description
IBM_EVENT_ID	A numeric value that is retrieved from IBM_FETCH_ID record. This value is a unique ID for the event.
IBM_OBJECT_NAME	The name of the corresponding business graph.
IBM_OBJECT_KEYS	The get key property names in the Component Interface, followed by the key values in name-value pairs. This information is used for the component's retrieval from the EIS.
IBM_EVENT_STATUS	If the event is ready to be polled, the status is set to 0 and the IBMPublishEvent function is called.
IBM_OBJECT_VERB	The verb that is set on the business object graph that contains the retrieved business object.
IBM_EVENT_DTTM	The date on which the event is created. For a future dated event, this is the effective date.
IBM_NEXT_EVENT_ID	The field that has the latest event ID under the record IBM_FETCH_ ID. This field is incremented for each event that is added to the IBM_EVENT_TBL, and it populates the IBM_EVENT_ID field in that table.
IBM_XID	The transaction ID that is needed to provide assured event delivery.

3. Create a record named IBM_EVENT_TBL and add to it all the fields that you have just created, except IBM_NEXT_EVENT_ID.
4. Create a record named IBM_FETCH_ID and add to it only the IBM_NEXT_EVENT_ID field.
5. Open the IBM_FETCH_ID record, select the IBM_NEXT_EVENT_ID field, view the PeopleCode, and select **fieldformula.**
6. Copy the PeopleCode for a custom event project from “PeopleCode for a custom event project” on page 1427 to the project that you are creating.
7. Create a page under your project that contains the fields of the IBM_EVENT_TBL record at level 0. The page can have any name.
8. Create a component under your project that contains the page that you have just created. The component can have any name.
9. Create a Component Interface against this component and give it any name. Confirm that you want to default the properties that are based on the underlying component definition.
10. Build the entire project, selecting all create options.
11. Test and confirm that the Component Interface works, by using the Component Interface tester.

12. Generate the Java APIs for the Component Interface, then add the generated classes to the adapter classpath. For complete information about building a PeopleTools project and testing the PeopleSoft Component Interface, refer to PeopleSoft documentation.

Connecting to an EIS by using the Adapter Connection wizard

Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

Before you start:

- Read "WebSphere Adapters nodes" on page 10
- Prepare the environment for WebSphere Adapters nodes
- Perform the preparatory tasks listed in "Developing message flow applications that use WebSphere Adapters" on page 286

A message flow application that uses one of the WebSphere Adapters requires the following resources:

- One or more message flows that contain one or more WebSphere Adapters nodes
- A message set that contains the XML Schema Definitions (XSD) for the business objects in the Enterprise Information System (EIS)
- An adapter component file for the WebSphere Adapter that is being used

The Adapter Connection wizard creates these resources automatically.

The following steps describe how to connect to an EIS.

1. Before you run the wizard, you need to gather the following information from the EIS.
   - **SAP**
     - SAP system user name
     - SAP system password
     - SAP host name or IP address
     - SAP Client ID (for example, 001)
     - SAP system number (for example, 00)
     - Language code (for example, EN)

     For more information, see "SAP connection properties for the Adapter Connection wizard" on page 1335.
   - **Siebel**
     - Siebel user name
     - Siebel password
     - Siebel host name or IP address
     - Language code

     For more information, see "Siebel connection properties for the Adapter Connection wizard" on page 1407.
   - **PeopleSoft**
     - PeopleSoft user name
     - PeopleSoft password
     - PeopleSoft host name or IP address
     - Port number (for example, 9000)

– Language code (for example, ENG)

For more information, see “PeopleSoft connection properties for the Adapter Connection wizard” on page 1431.

2. Switch to the Broker Application Development perspective.
3. Click **File** → **New** → **Adapter Connection**. The Adapter Connection wizard opens.
4. Follow the instructions in the wizard. To see a description of each field within the wizard, hold the mouse pointer over the field.

   Ensure that inbound and outbound SAP IDocs have different names if they are stored in the same message set. For more information, see An error is issued when you use the message set that is generated by the Adapter Connection wizard.

When you have completed the steps in the wizard, the specified message set project contains a message set with a message type for each business object that is to be used, and the specified message flow project references the message set project.

When the Adapter Connection wizard completes, the workbench displays a message that prompts you to drag the adapter component onto the message flow canvas.

1. Ensure that a message flow is open in the Message Flow editor so that the message flow canvas is available.
2. In the Broker Development view, expand the folders beneath the message set until you see the adapter component, which has a suffix of inadapter or outadapter.
3. Drag the adapter component onto the message flow canvas. The component appears as a message flow node.
4. Configure the node, as described in “Configuring a message flow node” on page 276.
5. When you have developed and saved your message flow, deploy it by following the instructions in Deploying a message flow application that uses WebSphere Adapters.

Using configurable services for SAP nodes

SAP nodes can get SAP connection details from either the adapter component or a configurable service. By using configurable services, you can change the connection details for adapters without the need to redeploy the adapters. For more details about creating, changing, reporting, and deleting the configurable services for SAP, see “Changing connection details for SAP adapters.”

Changing connection details for SAP adapters

SAP nodes can get SAP connection details from either the adapter component or a configurable service. By using configurable services, you can change the connection details for adapters without the need to redeploy the adapters. To pick up new values when a configurable service is created or modified, you must reload the broker or execution group to which the adapter was deployed, by using the mqsistop and mqsistart commands, or the mqsireload command.

Before you start:

- Read “WebSphere Adapters nodes” on page 10 and “Configurable services” on page 66 for background information.

Use the SAPConnection configurable service to change connection details for an SAP adapter. The SAP node reads all connection properties from the adapter component that it is configured to use. If a configurable service exists that has the same name as the node's adapter component, the node uses the values that are defined in that configurable service to override the corresponding properties from the adapter. If any properties on the configurable service are set to an empty string, the values that are configured in the `.inadapter` or `.outadapter` file are used. The properties of the SAP configurable services are described in Configurable services properties.

Creating, changing, reporting, and deleting configurable services

- To create a configurable service, use the mqsicreateconfigurableservice command, as shown in the following example. This example creates an SAPConnection configurable service for the SAP adapter *mySAPAdapter.outadapter* that connects to the SAP host `test.sap.ibm.com`, and uses client *001* for the connections into that server:

  ```
  mqsicreateconfigurableservice WBRK61_DEFAULT_BROKER -c SAPConnection
  -o mySAPAdapter.outadapter -n applicationServerHost,client
  -v test.sap.ibm.com,001
  ```

  To pick up the new values in the configurable service, restart the execution group and message flow.
- To change a configurable service, use the mqsichangeproperties command, as shown in the following example. This example changes the connections that are used by the adapter *mySAPAdapter.outadapter*. After you run this command, all adapters connect to the production system (`production.sap.ibm.com`) instead of the test system (`test.sap.ibm.com`):

  ```
  mqsichangeproperties WBRK61_BROKER -c SAPConnection -o mySAPAdapter.outadapter
  -n applicationServerHost -v production.sap.ibm.com
  ```

  To pick up the updated values in the configurable service, restart the execution group and message flow.
- To display all SAPConnection configurable services, use the mqsireportproperties command, as shown in the following example:

  ```
  mqsireportproperties WBRK61_DEFAULT_BROKER -c SAPConnection -o AllReportableEntityNames -r
  ```
- You can delete a configurable service that you have created by using the mqsideleteconfigurableservice command, as shown in the following example:

  ```
  mqsideleteconfigurableservice WBRK61_DEFAULT_BROKER -c SAPConnection
  -o mySAPAdapter.outadapter
  ```

Changing connection details for Siebel adapters

Siebel nodes can get Siebel connection details from either the adapter component or a configurable service. By using configurable services, you can change the connection details for adapters without the need to redeploy the adapters. To pick up new values when a configurable service is created or modified, you must reload the broker or execution group to which the adapter was deployed, by using the mqsistop and mqsistart commands, or the mqsireload command.

Before you start:

- Read “WebSphere Adapters nodes” on page 10 and “Configurable services” on page 66 for background information.

Use the SiebelConnection configurable service to change connection details for a Siebel adapter. The Siebel node reads all connection properties from the adapter component that it is configured to use. If a configurable service exists that has the same name as the node's adapter component, the node uses the values that are defined in that configurable service to override the corresponding properties from the adapter. If a configurable service is being used, all properties that are exposed by the configurable service are taken from the configurable service. The only properties that are taken from the adapter are those that you cannot set on the configurable service. The properties of the Siebel configurable service are described in Configurable services properties.

Creating, changing, reporting, and deleting configurable services

- To create a configurable service, use the mqsicreateconfigurableservice command, as shown in the following example. This example creates a SiebelConnection configurable service for the Siebel instance that is running on *my.siebel.qa.com*:

```
mqsicreateconfigurableservice WBRK61_DEFAULT_BROKER -c SiebelConnection -o mySiebelAdapter.outadapter
-n connectString -v "siebel://my.siebel.qa.com/SBA_80/SSEObjMgr_enu"
```

  To pick up the new values in the configurable service, restart the execution group and message flow.
- To change a configurable service, use the mqsichangeproperties command, as shown in the following example. This example changes the connections that are used by the adapter *mySiebelAdapter.outadapter*. After you run this command, all adapters connect to the production system (`my.siebel.production.com`) instead of the test system (`my.siebel.qa.com`):

```
mqsichangeproperties WBRK61_BROKER -c SiebelConnection -o mySiebelAdapter.outadapter -n connectString
-v "siebel://my.siebel.production.com/SBA_80/SSEObjMgr_enu"
```

  To pick up the updated values in the configurable service, restart the execution group and message flow.
- To display all SiebelConnection configurable services, use the mqsireportproperties command, as shown in the following example:

```
mqsireportproperties WBRK61_DEFAULT_BROKER -c SiebelConnection -o AllReportableEntityNames -r
```

- You can delete a configurable service that you have created by using the mqsideleteconfigurableservice command, as shown in the following example:

```
mqsideleteconfigurableservice WBRK61_DEFAULT_BROKER -c SiebelConnection -o mySiebelAdapter.outadapter
```

Changing connection details for PeopleSoft adapters

PeopleSoft nodes can get PeopleSoft connection details from either the adapter component or a configurable service. By using configurable services, you can change the connection details for adapters without the need to redeploy the adapters. To pick up new values when a configurable service is created or modified, you must reload the broker or execution group to which the adapter was deployed, by using the mqsistop and mqsistart commands, or the mqsireload command.

Before you start:

- Read "WebSphere Adapters nodes" on page 10 and "Configurable services" on page 66 for background information.

Use the PeopleSoftConnection configurable service to change connection details for a PeopleSoft adapter. The PeopleSoft node reads all connection properties from the adapter component that it is configured to use. If a configurable service exists that

has the same name as the node's adapter component, the node uses the values that are defined in that configurable service to override the corresponding properties from the adapter. If a configurable service is being used, all properties that are exposed by the configurable service are taken from the configurable service. The only properties that are taken from the adapter are those that you cannot set on the configurable service. The properties of the PeopleSoft configurable service are described in Configurable services properties.

Creating, changing, reporting, and deleting configurable services

- To create a configurable service, use the mqsicreateconfigurableservice command, as shown in the following example. This example creates a PeopleSoftConnection configurable service for the PeopleSoft instance that is running on *my.peoplesoft.qa.com*:

```
mqsicreateconfigurableservice WBRK61_DEFAULT_BROKER -c PeopleSoftConnection -o myPeopleSoftAdapter.outadapter
-n hostName,port -v "my.peoplesoft.qa.com",9000
```

  To pick up the new values in the configurable service, restart the execution group and message flow.
- To change a configurable service, use the mqsichangeproperties command, as shown in the following example. This example changes the connections that are used by the adapter *myPeopleSoftAdapter.outadapter*. After you run this command, all adapters connect to the production system (`my.peoplesoft.production.com`) instead of the test system (`my.peoplesoft.qa.com`):

```
mqsichangeproperties WBRK61_BROKER -c PeopleSoftConnection -o myPeopleSoftAdapter.outadapter -n hostName
-v "my.peoplesoft.production.com"
```

  To pick up the updated values in the configurable service, restart the execution group and message flow.
- To display all PeopleSoftConnection configurable services, use the mqsireportproperties command, as shown in the following example:

```
mqsireportproperties WBRK61_DEFAULT_BROKER -c PeopleSoftConnection -o AllReportableEntityNames -r
```

- You can delete a configurable service that you have created by using the mqsideleteconfigurableservice command, as shown in the following example:

```
mqsideleteconfigurableservice WBRK61_DEFAULT_BROKER -c PeopleSoftConnection -o myPeopleSoftAdapter.outadapter
```

Developing ESQL

Customize processing implemented by the Compute, Database, and Filter nodes in your message flows by coding ESQL.

You must create, for each node, an ESQL module in which you code the ESQL statements and functions to tailor the behavior of the node. You can access message content, or database content, or both, to achieve the results that you require. ESQL modules are maintained in ESQL files, managed through the Broker Application Development perspective.

This section provides information about:

- "ESQL overview" on page 303
- "Managing ESQL files" on page 313
- "Writing ESQL" on page 324

You can use the ESQL debugger, which is part of the flow debugger, to debug the code that you write. The debugger steps through ESQL code statement by statement, so that you can view and check the results of every line of code that is executed.

ESQL overview

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

This section contains introductory information about ESQL.

- For descriptions of ESQL user tasks, see "Writing ESQL" on page 324.
- For reference information about ESQL, see ESQL reference.

Read the following information before you proceed:

- An overview of message flows in "Message flows overview" on page 4.
- An overview of message trees in "The message tree" on page 69, and the topics within this container, paying special attention to "Logical tree structure" on page 76.

ESQL is based on Structured Query Language (SQL) which is in common usage with relational databases such as DB2. ESQL extends the constructs of the SQL language to provide support for you to work with message and database content to define the behavior of nodes in a message flow.

The ESQL code that you create to customize nodes within a message flow is defined in an ESQL file, typically named `<message_flow_name>.esql`,, which is associated with the message flow project. You can use ESQL in the following built-in nodes:

- "Compute node" on page 894
- "Database node" on page 902
- "Filter node" on page 970

You can also use ESQL to create functions and procedures that you can use in the following built-in nodes:

- "DataDelete node" on page 923
- "DataInsert node" on page 926
- "DataUpdate node" on page 930
- "Extract node" on page 944
- "Mapping node" on page 1052
- "Warehouse node" on page 1307

To use ESQL correctly and efficiently in your message flows, you must also understand the following concepts:

- Data types
- Variables
- Field references
- Operators
- Statements
- Functions
- Procedures
- Modules

Use the ESQL debugger, which is part of the flow debugger, to debug the code that you write. The debugger steps through ESQL code statement by statement, so that you can view and check the results of every line of code that is executed.

ESQL data types

A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data

that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

Within a broker, the fields of a message contain data that has a definite data type. It is also possible to use intermediate variables to help process a message. You must declare all such variables with a data type before use. A variable's data type is fixed; If you try to assign values of a different type you get either an implicit cast or an exception. Message fields do not have a fixed data type, and you can assign values of a different type. The field adopts the new value and type.

It is not always possible to predict the data type that results from evaluating an expression. This is because expressions are compiled without reference to any kind of message schema, and so some type errors are not caught until run time.

ESQL defines the following categories of data. Each category contains one or more data types.
- Boolean
- Datetime
- Null
- Numeric
- Reference
- String

ESQL variables

An ESQL variable is a data field that is used to help process a message.

You must declare a variable and state its type before you can use it. A variable's data type is fixed; if you code ESQL that assigns a value of a different type, either an implicit cast to the data type of the target is implemented or an exception is raised (if the implicit cast is not supported).

To define a variable and give it a name, use the DECLARE statement.

The names of ESQL variables are case sensitive; therefore, make sure that you use the correct case in all places. The simplest way to guarantee that you are using the correct case is always to define variables using uppercase names.

The workbench marks variables that have not been defined. Remove all these warnings before deploying a message flow.

You can assign an initial value to the variable on the DECLARE statement. If an initial value is not specified, scalar variables are initialized with the special value NULL, and ROW variables are initialized to an empty state. Subsequently, you can change the variable's value using the SET statement.

Three types of built-in node can contain ESQL code and therefore support the use of ESQL variables:
- “Compute node” on page 894
- “Database node” on page 902
- “Filter node” on page 970

Variable scope, lifetime, and sharing

How widespread and for how long a particular ESQL variable is available, is described by its scope, lifetime, and sharing:

A variable's scope
is a measure of the range over which it is visible. In the broker environment, the scope of variables is typically limited to the individual node.

A variable's lifetime
is a measure of the time for which it retains its value. In the broker environment, the lifetime of a variable varies but is typically restricted to the life of a thread within a node.

A variable's sharing characteristics
indicate whether each thread has its own copy of the variable or one variable is shared between many threads. In the broker environment, variables are typically not shared.

Types of variable

External
External variables (defined with the EXTERNAL keyword) are also known as *user-defined properties*, see "User-defined properties in ESQL." They exist for the entire lifetime of a message flow and are visible to all messages passing through the flow. You can define *external variables* only at the module and schema level. You can modify their initial values (optionally set by the DECLARE statement) at design time, using the Message Flow editor, or at deployment time, using the Broker Archive editor. You can query and set the values of user-defined properties at run time by using the Configuration Manager Proxy (CMP). For more information, see Setting user-defined properties at run time in a CMP application.

Normal
Normal variables variables have a lifetime of just one message passing through a node. They are visible to that message only. To define a *normal variables*, omit both the EXTERNAL and SHARED keywords.

Shared
Shared variables can be used to implement an in-memory cache in the message flow, see "Optimizing message flow response times" on page 197. *Shared variables* have a long lifetime and are visible to multiple messages passing through a flow, see "Long-lived variables" on page 306. They exist for the lifetime of the execution group process, the lifetime of the flow or node, or the lifetime of the node's SQL that declares the variable (whichever is the shortest). They are initialized when the first message passes through the flow or node after each broker startup.

See also the ATOMIC option of the BEGIN ... END statement. The BEGIN ATOMIC construct is useful when a number of changes need to be made to a shared variable and it is important to prevent other instances seeing the intermediate states of the data.

For information about specific types of variable, see:
- "User-defined properties in ESQL" (*external variables*)
- "Long-lived variables" on page 306 (*shared variables*)

User-defined properties in ESQL:

You can access user-defined properties (UDPs) as variables in your ESQL program by specifying the EXTERNAL keyword on a DECLARE statement. For example, the ESQL statement `DECLARE today EXTERNAL CHARACTER 'monday'` defines a user-defined property called today with an initial value monday.

Before you can use a user-defined property, you must define the property in the Message Flow editor when you construct a message flow that uses it. When you define a UDP in the Message Flow editor, you must define a value and the property type. The value can be a default value, which varies according to the type of the UDP. The value that is assigned to the UDP in the Message Flow editor takes precedence over a value that you have assigned to the UDP in your ESQL program.

you can also define a UDP for a subflow. A UDP has global scope and is not specific to a particular subflow. If you reuse a subflow in a message flow, and those subflows have identical UDPs, you cannot set the UDPs to different values.

Before you deploy the message flow that uses the UDP, you can change the value of the UDP in the Broker Archive editor. If you try to deploy a message flow that contains a UDP that has had no value assigned to it, a deployment failure occurs. For more information, see "Configuring a message flow at deployment time with user-defined properties" on page 454.

You can use UDPs to set configuration data, and use them like typical properties. No external calls to user-written plug-ins or parsing of environment trees are involved, and parsing costs of reading data out of trees are removed. The value of the UDP is finalized in the variable at deployment time.

You can declare UDPs only in modules or schemas. You can query, discover, and set UDPs at run time, to dynamically change the behavior of a message flow. For more information, see "User-defined properties" on page 134.

You can access UDPs from the following built-in nodes that use ESQL:
- Compute
- Database
- Filter

For a description of how to access a UDP from a JavaCompute node, see "Accessing message flow user-defined properties from a JavaCompute node" on page 540.

Long-lived variables:

You can use appropriate long-lived ESQL data types to cache data in memory.

Sometimes data has to be stored beyond the lifetime of a single message passing through a flow. One way to store this data is to store the data in a database. Using a database is good for long-term persistence and transactionality, but access (particularly write access) is slow.

Alternatively, you can use appropriate long-lived ESQL data types to provide an in-memory cache of the data for a certain period of time. Using long-lived ESQL data types makes access faster than from a database, although this speed is at the expense of shorter persistence and no transactionality.

You create long-lifetime variables by using the SHARED keyword on the DECLARE statement. For further information, see DECLARE statement.

The following sample demonstrates how to define shared variables using the DECLARE statement. The sample demonstrates how to store routing information

in a database table and use shared variables to store the database table in memory in the message flow to improve performance.

- Message Routing

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

Long-lived data types have an extended lifetime beyond that of a single message passing through a node. Long-lived data types are shared between threads and exist for the life of a message flow (strictly speaking the time between configuration changes to a message flow), as described in the following table.

	Scope	Life	Shared
Short lifetime variables			
Schema & Module	Node	Thread within node	Not at all
Routine Local	Node	Thread within routine	Not at all
Block Local	Node	Thread within block	Not at all
Long lifetime variables			
Node Shared	Node	Life of node	All threads of flow
Flow Shared	Flow	Life of flow	All threads of flow

Features of long-lived ESQL data types include:

- The ability to handle large amounts of long-lifetime data.
- The joining of data to messages fast.
- On multiple processor machines, multiple threads can access the same data simultaneously.
- Subsequent messages can access the data left by a previous message.
- Long lifetime read-write data can be shared between threads, because there is no long-term association between threads and messages.
- In contrast to data stored in database tables in the environment, this type of data is stored privately; that is, within the broker.
- ROW variables can be used to create a modifiable copy of the input message; see ESQL ROW data type.
- It is possible to create shared constants.

A typical use of these data types might be in a flow in which data tables are 'read-only' as far as the flow is concerned. Although the table data is not actually static, the flow does not change it, and thousands of messages pass through the flow before there is any change to the table data.

Examples include:

- A table which contains a day's credit card transactions. The table is created each day and that day's messages are run against it. Then the flow is stopped, the table updated and the next day's messages run. These flows might perform better if they cache the table data rather than read it from a database for each message.
- The accumulation and integration of data from multiple messages.

Broker properties

For each broker, WebSphere Message Broker maintains a set of properties. You can access some of these properties from your ESQL programs. A subset of the properties is also accessible from Java programs. It can be useful, at run time, to have real-time access to details of a specific node, flow, or broker.

Broker properties are divided into four categories:

- Properties that relate to a specific node
- Properties that relate to nodes in general
- Properties that relate to a message flow
- Properties that relate to the execution group

For a description of the broker, flow, and node properties that are accessible from ESQL and Java, see Broker properties that are accessible from ESQL and Java.

Broker properties have the following characteristics.

- They are grouped by broker, execution group, flow, and node.
- They are case sensitive. Their names always start with an uppercase letter.
- They return NULL if they do not contain a value.

All nodes that allow user programs to edit ESQL support access to broker properties. These nodes are:

- Compute nodes
- Database nodes
- Filter nodes
- All derivatives of these nodes

User-defined properties can be queried, discovered, and set at run time to dynamically change the behavior of a message flow. You can use the Configuration Manager Proxy (CMP) API to manipulate these properties, which can be used by a systems monitoring tool to perform automated actions in response to situations that it detects in the monitored systems. For more information, see “User-defined properties” on page 134.

A *complex property* is a property to which you can assign multiple values. Complex properties are displayed in a table in the Properties view, where you can add, edit, and delete values, and change the order of the values in the table. You cannot promote complex properties; therefore, they do not appear in the Promote properties dialog box. Nor can you configure complex properties; therefore, they are not supported in the Broker Archive editor. For an example of a complex property, see the Query elements property of the DatabaseRoute node.

For more information about editing a node's properties, see “Configuring a message flow node” on page 276.

ESQL field references

An ESQL field reference is a sequence of period-separated values that identify a specific field (which might be a structure) within a message tree or a database table.

The path from the root of the information to the specific field is traced using the parent-child relationships.

A field reference is used in an ESQL statement to identify the field that is to be referenced, updated, or created within the message or database table. For example, you might use the following identifier as a message field reference:

```
Body.Invoice.Payment
```

You can use an ESQL variable of type REFERENCE to set up a dynamic pointer to contain a field reference. This might be useful in creating a fixed reference to a commonly-referenced point within a message; for example the start of a particular structure that contains repeating fields.

A field reference can also specify element types, XML namespace identifications, indexes and a type constraint; see ESQL field reference overview for further details.

The first name in a field reference is sometimes known as a *Correlation name*.

ESQL operators

An ESQL operator is a character or symbol that you can use in expressions to specify relationships between fields or values.

ESQL supports the following groups of operators:

- Comparison operators, to compare one value to another value (for example, less than). Refer to ESQL simple comparison operators and ESQL complex comparison operators for details of the supported operators and their use.
- Logical operators, to perform logical operations on one or two terms (for example, AND). Refer to ESQL logical operators for details of the supported operators and their use.
- Numeric operators, to indicate operations on numeric data (for example, +). Refer to ESQL numeric operators for details of the supported operators and their use.

There are some restrictions on the application of some operators to data types; not all lead to a meaningful operation. These are documented where they apply to each operator.

Operators that return a Boolean value (TRUE or FALSE), for example the greater than operator, are also known as predicates.

ESQL statements

An ESQL statement is an instruction that represents a step in a sequence of actions or a set of declarations.

ESQL provides a large number of different statements that perform different types of operation. All ESQL statements start with a keyword that identifies the type of statement and end with a semicolon. An ESQL program consists of a number of statements that are processed in the order they are written.

As an example, consider the following ESQL program:

```
DECLARE x INTEGER;
SET x = 42;
```

This program consists of two statements. The first starts with the keyword DECLARE and ends at the first semicolon. The second statement starts with the keyword SET and ends at the second semicolon. These two statements are written

on separate lines and it is conventional (but not required) that they be so. You will notice that the language keywords are written in capital letters. This is also the convention but is not required; mixed case and lowercase are acceptable.

The first statement declares a variable called x of type INTEGER, that is, it reserves a space in the computer's memory large enough to hold an integer value and allows this space to be subsequently referred to in the program by the name x. The second statement sets the value of the variable x to 42. A number appearing in an ESQL program without decimal point and not within quotation marks is known as an integer literal.

ESQL has a number of data types and each has its own way of writing literal values. These are described in "ESQL data types" on page 303.

For a full description of all the ESQL statements, see ESQL statements.

ESQL nested statements:

An ESQL nested statement is a statement that is contained within another statement.

Consider the following ESQL program fragment:

```
IF Size> 100.00 THEN
  SET X = 0;
  SET Y = 0;
  SET REVERSE = FALSE;
ELSE
  SET X = 639;
  SET Y = 479;
  SET REVERSE = TRUE;
END IF;
```

In this example, you can see a single IF statement containing the optional ELSE clause. Both the IF and ELSE portions contain three nested statements. Those within the IF clause are processed if the operator > (greater than) returns the value TRUE (that is, if Size has a value greater than 100.00); otherwise, those within the ELSE clause are processed.

Many statements can have expressions nested within them, but only a few can have statements nested within them. The key difference between an expression and a statement is that an expression calculates a value to be used, whereas a statement performs an action (usually changing the state of the program) but does not produce a value.

ESQL functions

A function is an ESQL construct that calculates a value from a number of given input values.

A function usually has input parameters and can, but does not usually have, output parameters. It returns a value calculated by the algorithm described by its statement. This statement is usually a compound statement, such as BEGIN... END, because this allows an unlimited number of nested statements to be used to implement the algorithm.

ESQL provides a number of predefined, or "built-in", functions which you can use freely within expressions. You can also use the CREATE FUNCTION statement to define your own functions.

When you define a function, you must give it a unique name. The name is handled in a case insensitive way (that is, use of the name with any combination of uppercase and lowercase letters matches the declaration). This is in contrast to the names that you declare for schemas, constants, variables, and labels, which are handled in a case sensitive way, and which you must specify exactly as you declared them.

Consider the following ESQL program fragment:

```
SET Diameter = SQRT(Area / 3.142) * 2;
```

In this example, the function SQRT (square root) is given the value inside the brackets (itself the result of an expression, a divide operation) and its result is used in a further expression, a multiply operation. Its return value is assigned to the variable Diameter. See Calling ESQL functions for information about all the built-in ESQL functions.

In addition, an ESQL expression can refer to a function in another broker schema (that is, a function defined by a CREATE FUNCTION statement in an ESQL file in the same or in a different dependent project). To resolve the name of the called function, you must do one of the following:

- Specify the fully-qualified name (<SchemaName>.<FunctionName>) of the called function.
- Include a PATH statement to make all functions from the named schema visible. Note that this technique only works if the schemas do not contain identically-named functions. The PATH statement must be coded in the same ESQL file, but not within any MODULE.

Note that you cannot define a function within an EVAL statement or an EVAL function.

ESQL procedures

An ESQL procedure is a subroutine that has no return value. It can accept input parameters from, and return output parameters to, the caller.

Procedures are very similar to functions. The main difference between them is that, unlike functions, procedures have no return value. Thus they cannot form part of an expression and are invoked by using the CALL statement. Procedures commonly have output parameters

You can implement a procedure in ESQL (an internal procedure) or as a database stored procedure (an external procedure). The ESQL procedure must be a single ESQL statement, although that statement can be a compound statement such as BEGIN END. You cannot define a procedure within an EVAL statement or an EVAL function.

When you define a procedure, give it a name. The name is handled in a case insensitive way (that is, use of the name with any combination of uppercase and lowercase letters matches the declaration). That is in contrast to the names that you declare for schemas, constants, variables, and labels, which are handled in a case sensitive way, and which you must specify exactly as you declared them.

An ESQL expression can include a reference to a procedure in another broker schema (defined in an ESQL file in the same or a different dependent project). If you want to use this technique, either fully qualify the procedure, or include a

PATH statement that sets the qualifier. The PATH statement must be coded in the same ESQL file, but not within a MODULE.

An external database procedure is indicated by the keyword EXTERNAL and the external procedure name. This procedure must be defined in the database and in the broker, and the name specified with the EXTERNAL keyword and the name of the stored database procedure must be the same, although parameter names do not have to match. The ESQL procedure name can be different from the external name it defines.

Overloaded procedures are not supported to any database. (An overloaded procedure is one that has the same name as another procedure in the same database schema which has a different number of parameters, or parameters with different types.) If the broker detects that a procedure has been overloaded, it raises an exception.

Dynamic schema name resolution for stored procedures is supported; when you define the procedure you must specify a wildcard for the schema that is resolved before invocation of the procedure by ESQL. This is explained further in "Invoking stored procedures" on page 376.

ESQL modules

A module is a sequence of declarations that define variables and their initialization, and a sequence of subroutine (function and procedure) declarations that define a specific behavior for a message flow node.

A module must begin with the `CREATE node_type MODULE` statement and end with an `END MODULE` statement. The node_type must be one of COMPUTE, DATABASE, or FILTER. The entry point of the ESQL code is the function named MAIN, which has MODULE scope.

Each module is identified by a name which follows `CREATE node_type MODULE`. The name might be created for you with a default value, which you can modify, or you can create it yourself. The name is handled in a case insensitive way (that is, use of the name with any combination of uppercase and lowercase letters matches the declaration). That is in contrast to the names that you declare for schemas, constants, variables, and labels, which are handled in a case sensitive way, and which you must specify exactly as you declared them.

You must create the code for a module in an ESQL file which has a suffix of `.esql`. You must create this file in the same broker schema as the node that references it. There must be one module of the correct type for each corresponding node, and it is specific to that node and cannot be used by any other node.

When you create an ESQL file (or complete a task that creates one), you indicate the message flow project and broker schema with which the file is associated as well as specifying the name for the file.

Within the ESQL file, the name of each module is determined by the value of the corresponding property of the message flow node. For example, the property *ESQL Module* for the Compute node specifies the name of the node's module in the ESQL file. The default value for this property is the name of the node. You can specify a different name, but you must ensure that the value of the property and the name of the module that provides the required function are the same.

The module must contain the function MAIN, which is the entry point for the module. This is included automatically if the module is created for you. Within MAIN, you can code ESQL to configure the behavior of the node. If you include ESQL within the module that declares variables, constants, functions, and procedures, these are of local scope only and can be used within this single module.

If you want to reuse ESQL constants, functions, or procedures, you must declare them at broker schema level. You can then refer to these from any resource within that broker schema, in the same or another project. If you want to use this technique, either fully qualify the procedure, or include a PATH statement that sets the qualifier. The PATH statement must be coded in the same ESQL file, but not within any MODULE.

Managing ESQL files

Within a message flow project, you can create ESQL files to contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, or Filter nodes.

The ESQL code is contained within a module that is associated with the node. Each module must be created within an ESQL file. The name of the module within the ESQL file must match the name specified for the module in the *ESQL Module* property of the corresponding node. Although you can modify the module name, and change it from its default value (which is the name of the message flow, concatenated with the name of the node with which the module is associated), ensure that the module in the ESQL file matches the node property.

The following topics describe how you can manage these files:

Creating an ESQL file

When you include a node in your message flow that requires ESQL to customize its function (the Compute, Database, and Filter nodes), you must code the ESQL statements that provide the customization in an ESQL module within an ESQL file. You can use the same ESQL file for more than one module.

Before you start

To complete this task, you must have completed the following task:

- "Creating a message flow project" on page 256

ESQL files are stored in a file system or in a shared repository. If you are using a file system, this can be the local file system or a shared drive. If you store files in a repository, you can use any of the available repositories that are supported by Eclipse, for example CVS.

To create an ESQL file:

1. Switch to the Broker Application Development perspective.
2. Click **File** → **New** → **Message Flow ESQL File**.

   You can also press Ctrl+N. This displays a dialog box that allows you to select the wizard to create a new object. Click Message Brokers in the left view; the right view displays a list of objects that you can create for WebSphere Message Broker. Click Message Flow ESQL File in the right view, then click **Next**. The New Message Flow ESQL File wizard is displayed.
3. Enter the name of the message flow project in which to create the ESQL file. You must enter the name of an existing message flow project. The dialog box is displayed with the current project name entered in the project name field. You can accept this value or change it to specify a different project. You can also click **Browse** to view a list of valid projects (projects that are defined and displayed in the Navigator view), and select the appropriate value from that list. The list is filtered to only show projects in the active working set.

   If you type in the name of a project that does not exist, the error message `The specified project does not exist` is displayed in the dialog box and you cannot continue until you specify a valid project name.
4. If you want the ESQL file to be defined within a specific broker schema, enter the name of the broker schema in the appropriate entry field, or click **Browse** to select the broker schema from the list of valid broker schema for this project. (If only the default broker schema is defined in this project, **Browse** is disabled.)
5. Enter a name for the new ESQL file. If you enter a name that is already in use for an ESQL file in this project, the error message `The resource <name>.esql already exists` is displayed in the dialog box and you cannot continue until you specify a valid name.

   When creating ESQL files, the overall file path length must not exceed 256 characters, due to a Windows file system limitation. If you try to add a message flow to a broker archive file with ESQL or mapping files with a path length that exceeds 256 characters, the compiled message flow will not be generated and cannot be deployed. Therefore, make sure that the names of your ESQL files, mapping files, projects, and broker schema are as short as possible.

An ESQL file can also be created automatically for you. If you select Open ESQL from the menu displayed when you right-click a Compute, Database, or Filter node, and the module identified by the appropriate property does not already exist within the broker schema, a module is automatically created for you. This is created in the file `<message_flow_name>.esql` in the same broker schema within the same project as the `<message_flow_name>.msgflow` file. If that ESQL file does not already exist, that is also created for you.

The contents of a single ESQL file do not have any specific relationship with message flows and nodes. It is your decision which modules are created in which files (unless the specified module, identified by the appropriate property, is created by default in the file `<message_flow_name>.esql` as described above). Monitor the size and complexity of the ESQL within each file, and split the file if it becomes difficult to view or manage.

If you create reusable subroutines (at broker schema level) within an ESQL file, you might want to refer to these routines from ESQL modules in another project. To do this, specify that the project that wants to invoke the subroutines depends on the project in which the ESQL file containing them is defined. You can specify

this when you create the second project, or you can update project dependencies by selecting the project, clicking **Properties**, and updating the dependencies in the Project Reference page of the Properties dialog box.

Opening an existing ESQL file

You can add to and modify ESQL code that you have created in an ESQL file in a message flow project.

Before you start

To complete this task, you must have completed the following task:
- "Creating an ESQL file" on page 313

To open an existing ESQL file:

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, double-click the ESQL file that you want to open. The file is opened in the editor view.
3. Work with the contents of file to make your changes. The file can contain modules relating to specific nodes in a message flow, PATH statements, and declarations at broker schema level such as reusable constants and procedures. Scroll through the file to find the specific content that you want to work with.
4. You can select the content that you want to work with by selecting its name in the Outline view. The code for the selected resource is highlighted.

You can also open an ESQL file when you have a message flow open in the editor view by selecting an appropriate node (of type Compute, Database, or Filter), right-clicking, and selecting **Open ESQL**. In this case, the ESQL file that contains this module is opened, and the module for the selected node is highlighted in the editor view.

Creating ESQL for a node

Create ESQL to customize the behavior of a Compute, Database, or Filter node within an ESQL file.

Before you start

Complete the following task:
- "Creating an ESQL file" on page 313

Within the ESQL file, create a module that is associated with a node in your message flow. A module can be associated with only one node of a particular type (Compute, Database, or Filter). Within the module you can create and use functions and procedures as well as the supplied statements and functions. You can also create local constants and variables.

If you have created constants, functions, or procedures at the broker schema level, you can also refer to these within the module. You can define routines at a level at which many different modules can use them, which can save you development time and maintenance effort.

To create ESQL for a node:

1. Switch to the Broker Application Development perspective.

2. In the Broker Development view, double-click the message flow that includes the node for which you want to create ESQL. The message flow opens in the editor view.
3. Right-click the node (which must be Compute, Database, or Filter) and then click **Open ESQL**. The default ESQL file for this message flow, *message_flow_name*.esql, is opened in the editor view. The file is created if it does not already exist.

   If you have already created the file, it is opened in the editor view and a new module is created and highlighted. If the file is created for you, it contains a skeleton module for this node at the end. Its exact content depends on the type of node.

   The following module is created for a Compute node:

```
CREATE COMPUTE MODULE module_name
       CREATE FUNCTION Main() RETURNS BOOLEAN
       BEGIN
              -- CALL CopyMessageHeaders();
              -- CALL CopyEntireMessage();
              RETURN TRUE;
       END;

       CREATE PROCEDURE CopyMessageHeaders() BEGIN
              DECLARE I INTEGER 1;
              DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
              WHILE I < J DO
                     SET OutputRoot.*[I] = InputRoot.*[I];
                     SET I = I + 1;
              END WHILE;
       END;

       CREATE PROCEDURE CopyEntireMessage() BEGIN
              SET OutputRoot = InputRoot;
       END;
END MODULE;
```

   The module name is determined by the value that you have set for the corresponding node property. The default is *message_flow_name_node_type*. The Main function contains calls to two procedures, described in the following list, that are declared within the Compute node module following the function Main. These calls are commented out. If you want to include the function that they provide, uncomment the lines and place them at the appropriate point in the ESQL that you create for Main.

   CopyMessageHeaders
   : This procedure loops through the headers contained in the input message and copies each one to the output message.

   CopyEntireMessage
   : This procedure copies the entire contents of the input message, including the headers, to the output message.

   If you create an ESQL module for a Database node, the following module is created:

```
CREATE DATABASE MODULE module_name
       CREATE FUNCTION Main() RETURNS BOOLEAN
       BEGIN
               RETURN TRUE;
       END;
END MODULE;
```

For a Filter node, the module is identical to that created for the Database node except for the first line, which reads:

```
CREATE FILTER MODULE module_name
```

4. Add ESQL to this file to customize the behavior of the node.

   Start by adding ESQL statements within the Main function, that is after the BEGIN statement, and before RETURN TRUE. You can add DECLARE statements within the module that are not within the Main function. To add a new line into the file, press Enter.

   To help you to code valid ESQL, the editor displays a list of valid statements and functions at the point of the cursor. To invoke this assistance, click **Edit** → **Content Assist**. On some systems, you can use the key combination Ctrl+Space. Scroll through the list displayed to find and highlight the one that you want, and press Enter. The appropriate code is inserted into your module, and the list disappears.

   Content assistance is provided in the following areas:

   - Applicable keywords, based on language syntax.
   - Blocks of code that go together, such as `BEGIN END;`.
   - Constants that you have defined, identifiers, labels, functions, and procedures that can be used, where the routines can be in any projects, even if the current project does not reference them.
   - Database schema and table names after the database correlation name, as well as table column names in INSERT, UPDATE, DELETE, and SELECT statements, and, in most cases, the WHERE clauses of those statements.
   - Elements of message field reference: runtime domain (parser) names, format of type expression, namespace identifiers, namespace-qualified element and attribute names, and format of index expression.
   - Content in the Properties folder under the output message root.
   - For the DECLARE NAMESPACE statement, target namespaces of message sets and schema names.

   Content assistance works only if the ESQL can be parsed correctly. Errors such as END missing after BEGIN, and other unterminated block statements, cause parser failures and no content assistance is provided. Try content assistance in other areas around the statement where it does not work to narrow down the point of error. Alternatively, save the ESQL file; saving the file causes validation, and all syntax errors are written to the Problems view. Refer to the errors reported to understand and correct the ESQL syntax. If you use content assistance to generate most statements (such as block statements), these are correctly entered and there is less opportunity for error.
5. When you have finished working with this module, you can close the ESQL file. Save the file before you close it to retain all your changes and validate your ESQL.

If you prefer, you can open the ESQL file directly and create the module within that file using the editor:

1. Switch to the Broker Application Development perspective.
2. Select the ESQL file in which you want to create the module. Either double-click to open this file in the editor view, or right-click and click **Open**.
3. In the editor view, position your cursor on a new line and use content assistance to select the appropriate module skeleton for this type of node, for example `CREATE COMPUTE MODULE END MODULE;`. You can type this text in yourself

if you prefer, but you must ensure that what you type is consistent with the required skeleton, shown earlier. Use content assistance to give you additional help by inserting only valid ESQL, and by inserting matching end statements (for example, `END MODULE;`) where these are required.

4. Complete the coding of the module as appropriate.

Whichever method you use to open the ESQL file, be aware that the editor provides functions to help you to code ESQL. This section refers to content assistance; other functions are available. For information about these functions, see ESQL editor.

Modifying ESQL for a node

If you want to change the customization of a node that requires ESQL (Compute, Database, or Filter), you can modify the ESQL statements within the module that you created for that node.

Before you start

To complete this task, you must have completed the following task:
- "Creating ESQL for a node" on page 315

To modify ESQL code:

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, select the message flow that you want to work with and double-click it. The message flow is opened in the editor view.
3. Right-click the node corresponding to the ESQL module that you want to modify and click **Open ESQL**. The ESQL file is opened in the editor view. The module for this node is highlighted.
4. Make the changes that you want in the module, by entering new statements (remember that you can use Content Assist, available from the Edit menu or, on some systems, by pressing Ctrl+Space), changing existing statements by overtyping, or deleting statements using the Delete or backspace keys. Note that, to get Content Assist to work with message references, you must set up a project reference from the project containing the ESQL to the project containing the message set. For information about setting up a project reference, see Project references.
5. You can change the name of the module that you are working with, by over-typing the current name with the new one. Remember that, if you do that, you must also change the node property *ESQL Module* to reflect the new name to ensure that the correct ESQL code is deployed with the node.
6. When you have finished working with this module, you can close the ESQL file. Save the file before you close it to retain all your changes and validate your ESQL.

If you prefer, you can open the ESQL file directly by double-clicking it in the Broker Development view. You can select the module that you want to work with from the Outline view.

The editor provides functions that you can use to help you modify your ESQL code. These functions are described in ESQL editor.

You can also modify the ESQL source by selecting **Source** → **Format**. This option formats all selected lines of code (unless only partially selected, when they are ignored), or, if no lines are selected, formats the entire file (correcting alignments and indentation).

Adding comments to ESQL:

You can add comments to and remove comments from your ESQL code:

1. To change an existing line of code into a comment line, click **Source** → **Comment**.
2. To change a comment line to a code line, click **Source** → **Uncomment**.
3. To create a new comment line, press Enter to create a new line and either type the comment identifier **--** or click **Source** → **Comment**. You can enter any text after the identifier: everything you type is ignored by the ESQL editor.

Saving an ESQL file

When you edit your ESQL file, you can save it both to preserve the additions and modifications that you have made, and to force the editor to validate the content of the file.

Before you start

To complete this task, you must have completed the following task:

- "Creating an ESQL file" on page 313

To save an ESQL file:

1. Switch to the Broker Application Development perspective.
2. Create an ESQL file, or open an existing ESQL file.
3. Change the contents of the ESQL file.
4. When you have finished working, save the file to retain all your changes by clicking **File** → **Save** ***filename*****.esql** or **File** → **Save All** (the menu always shows the current filename correctly).

   When you save the file, the validator is called by the editor to check that the ESQL obeys all grammar and syntax rules (specified by the syntax diagrams and explanations in ESQL reference).

   You can request additional validation when you set ESQL preferences:

   a. Click **Window** → **Preferences**. The Preferences dialog is displayed.
   b. Expand the item for **Broker Development** on the left, expand **ESQL**, and click **Validation**.
   c. Select the level of validation that you require for each category of error:
      1) Unresolved identifiers
      2) Message references do not match message definitions
      3) Database references do not match database schema
      4) Use of deprecated keywords

      The default level is **Warning**; you can change this value to **Error** or **Ignore**.

      Validating message definitions can affect response times in the editor, particularly if you have complicated ESQL that makes many references to a complex message definition. You might choose to delay this validation. Call validation when you have finished developing the message flow and are about to deploy it, to avoid runtime errors. For each error found, the editor writes the code line number and the reason for the error; errors are created as entries in the Problems view.

5. If you double-click the error, the editor positions your cursor on the line in which it found that error. The line is also highlighted by the error icon  in the margin to the left.

   The editor might also find potential error situations, that it highlights as warnings (with the warning icon ); it also writes these warnings to the Problems view. For example, you might have included a `BROKER SCHEMA` statement that references an invalid schema (namespace).

   Check your code, and make the corrections required by that statement or function.

Save As:

You can save a copy of this ESQL file by using **File → Save As**.

1. Click **File → Save *name* As**.
2. Specify the message flow project in which you want to save a copy of the ESQL file. The project name defaults to the current project. You can accept this name, or choose another name from the valid options that are displayed in the File Save dialog.
3. Specify the name for the new copy of the ESQL file. If you want to save this ESQL file in the same project, you must either give it another name, or confirm that you want to overwrite the current copy (that is, copy the file to itself).

   If you want to save this ESQL file in another project, the project must exist (you can only select from the list of existing projects). You can save the file with the same or another name in another project.
4. Click **OK**. The message flow is saved and the message flow editor validates its contents. The editor provides a report of all errors that it finds in the Problems view.

Copying an ESQL file

You might find it useful to copy an ESQL file as a starting point for a new ESQL file that has similar function.

Before you start

To complete this task, you must have completed the following task:
- "Creating an ESQL file" on page 313

To copy an ESQL file:

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, select the ESQL file (`<message_flow_name>.esql`) that you want to copy. Right-click the file and click **Copy** from the menu.
3. Right-click the broker schema within the message flow project to which you want to copy the ESQL file and click **Paste**. You can copy the ESQL file to the same broker schema within the same message flow project, or to a different broker schema within the same message flow project, or to a broker schema in a different message flow project.

   When you copy an ESQL file, the associated files (message flow, and mapping if present) are not automatically copied to the same target message flow project. If you want these files copied as well, you must do this explicitly following this procedure.

If you want to use this ESQL file with another message flow, ensure that the modules within the ESQL file match the nodes that you have in the message flow, and that the node properties are set correctly.

You can also use **File** → **Save As** to copy an ESQL file. This is described in "Saving an ESQL file" on page 319.

Renaming an ESQL file

You can rename an ESQL file within the message flow project. You might want to do this, for example, if you have renamed the message flow that it is associated with.

Before you start

To complete this task, you must have completed the following task:
- "Creating an ESQL file" on page 313

To rename an ESQL file:

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the ESQL file that you want to rename. Its default name is `<message_flow_name>.esql`. Click **Rename** or click **File** → **Rename**. If you have selected the ESQL file, you can press F2. The Rename Resource dialog is displayed.
3. Enter the new name for the ESQL file. Click **OK** to complete the action, or **Cancel** to cancel the request. If you click **OK**, the ESQL file is renamed.

Moving an ESQL file

If you move a message flow from one broker schema to another, or from one project to another, you might want to move all ESQL files that are associated with that message flow.

Before you start

To complete this task, you must have completed the following task:
- "Creating an ESQL file" on page 313

To move an ESQL file:

1. Switch to the Broker Application Development perspective.
2. Move the ESQL file in one of the following ways:
   - Drag the ESQL file that you want to move from its current location to a broker schema within the same or another message flow project.

     If the target location that you have chosen is not valid (for example, if an ESQL file of this name exists in the broker schema), the invalid icon is displayed and the move is not completed.
   - Right-click the ESQL file and click **Move**, or click **File** → **Move**. The Move dialog is displayed.

     Select the project and the broker schema from the list of valid targets that is shown in the dialog.

     Click **OK** to complete the move, or **Cancel** to cancel the request.

     If you click **OK**, the ESQL file is moved to its new location.
3. Check the Problems view for errors (indicated by the error icon) or warnings (indicated by the warning icon) generated by the move.

The errors in the Problems view include those errors caused by broken references. When the move is completed, all references to this ESQL file are checked. If you have moved the file within the same named broker schema within the same message flow project, all references are still valid. If you have moved the file to another broker schema in the same or another message flow project, the references are broken. If you have moved the file to the same named broker schema in another message flow project, the references might be broken if the project references are not set correctly to recognize external references in this file. These errors occur because resources are linked by a fully qualified name.

4. Double-click each error or warning to open the message flow that has the error in the editor view, and highlight the node in error. You can now correct the error.

When you move an ESQL file, its associated files (for example, the message flow file) are not automatically moved to the same target broker schema. You must move these files yourself.

Changing ESQL preferences

You can modify the way in which ESQL is displayed in the editor and validated by the editor:

- "Changing ESQL editor settings"
- "Changing ESQL validation settings" on page 323

Changing ESQL editor settings:

When you open an ESQL file in the editor view, you can tailor the editor appearance by changing editor settings.

1. **To change ESQL editor settings:**
   a. Switch to the Broker Application Development perspective.
   b. Click **Window** → **Preferences**. The Preferences dialog box opens.
   c. Expand the item for ESQL on the left and click **ESQL Editor**.
   d. Update the settings available for tab width and colors:
      - Click the **General** tab to change the displayed tab width within the ESQL editor.
      - Click the **Colors** tab to change the color of the editor view background, and of the entities displayed in the editor view. These include comments and keywords in your ESQL code.
   e. When you have completed your changes, click either **Apply** (to apply your changes and leave the Preferences dialog box open), or **OK** (to apply your changes and close the dialog box). Alternatively, to close the dialog box and discard your changes, click **Cancel**.
   f. To return the ESQL editor settings to the initial values, click **Restore Defaults**. All values are reset to the original settings.

      If you change the editor settings when you have an editor session active, the changes are implemented immediately. If you do not have an editor session open, you see the changes when you next edit an ESQL file.
2. **To change font settings for the ESQL editor:**
   a. Click **Window** → **Preferences**. The Preferences dialog box opens.
   b. Expand the item for Workbench on the left of the Preferences dialog box, and click **Colors and Fonts**.
   c. On the **Colors and Fonts** tab, expand Basic.

d. Select a font or text color option and click **Change**. The Font dialog box opens.
e. When you have completed your changes, click either **Apply** (to apply your changes and leave the Preferences dialog box open) or **OK** (to apply your changes and close the dialog box). Alternatively, to close the dialog box and discard your changes, click **Cancel**.
f. To return the ESQL editor settings to the initial values, click **Restore Defaults**.

Changing ESQL validation settings:

You can specify the level of validation that the ESQL editor performs when you save a `.esql` file. If the validation you have requested results in warnings, you can deploy a BAR file containing this message flow. However, if errors are reported, you cannot deploy the BAR file.

To change ESQL validation settings:

1. Switch to the Broker Application Development perspective.
2. Click **Window** → **Preferences**. The Preferences dialog is displayed.
3. Expand the item for ESQL on the left and click Validation.
4. Update the settings for what is validated, and for what warnings or errors are reported. See ESQL editor for details of the settings and their values.
5. When you have completed your changes, click **Apply** to close the Preferences dialog, apply your changes and leave the Preferences dialog open. Click **OK** to apply your changes and close the dialog. Click **Cancel** to close the dialog and discard your changes.
6. If you want to return your ESQL editor preferences to the initial values, click **Restore Defaults**. All values are reset to the original settings.

If you make changes to the validation settings, the changes are implemented immediately for currently open edit sessions and for subsequent edit sessions.

Deleting ESQL for a node

If you delete a node from a message flow, you can delete the ESQL module that you created to customize its function.

Before you start

To complete this task, you must have completed the following task:
- "Creating ESQL for a node" on page 315

To delete ESQL code:

1. Switch to the Broker Application Development perspective.
2. Open the message flow that you want to work with by double-clicking it in the Broker Development view. The message flow is opened in the editor view.
3. Select the node for which you want to delete the ESQL module, right-click and click **Open ESQL**. The ESQL file is opened in the editor view, with the module for this node highlighted.
4. Press the Delete or backspace key to delete the whole module.
5. When you have finished working with this module, you can close the ESQL file. Save the file before you close it to retain all your changes. Save also validates your ESQL: see "Saving an ESQL file" on page 319.

If you prefer, you can open the ESQL file directly by double-clicking it in the Broker Development view. The ESQL file is opened in the editor view. Select the module that you want to delete from the Outline view and delete it as described above. You can also right-click on the module name in the Broker Development view (the modules in the ESQL file are visible if you expand the view of the file by clicking the + beside the file name) and click **Delete**.

Deleting an ESQL file

If you delete a message flow, or if you have deleted all the ESQL code in an ESQL file, you can delete the ESQL file.

Before you start

To complete this task, you must have completed the following task:
- "Creating an ESQL file" on page 313

To delete an ESQL file:

1. Switch to the Broker Application Development perspective.
2. Within the Broker Development view, right-click the ESQL file that you want to delete, and click **Delete**. A dialog is displayed that asks you to confirm the deletion.

   You can also select the file in the Broker Development view, and click **Edit** → **Delete**. A dialog is displayed that asks you to confirm the deletion.
3. Click **Yes** to delete the file, or **No** to cancel the delete request.

   If you maintain resources in a shared repository, a copy is retained in that repository. You can follow the instructions provided by the repository supplier to retrieve the file if required.

   If you are using the local file system or a shared file system to store your resources, no copy of the file is retained. Be careful to select the correct file when you complete this task.

Writing ESQL

How you can use ESQL to customize nodes.

When you create a message flow, you include input nodes that receive the messages and, optionally, output nodes that send out new or updated messages. If required by the processing that must be performed on the message, you can include other nodes after the input node that complete the actions that your applications need.

Some of the built-in nodes enable you to customize the processing that they provide. The Compute, Database, and Filter nodes require you to provide a minimum level of ESQL, and you can provide much more than the minimum to control precisely the behavior of each node. This set of topics discusses ESQL and the ways in which you can use it to customize these nodes.

The DataDelete, DataInsert, DataUpdate, Extract, Mapping, and Warehouse nodes provide a mapping interface with which you can customize their function. The ways in which you can use the mapping functions associated with these nodes are described in developing message mappings, see "Developing message mappings" on page 546.

ESQL provides a rich and flexible syntax for statements and functions that enable you to check and manipulate message and database content. You can:

- Read the contents of the input message
- Modify message content with data from databases
- Modify database content with data from messages
- Construct new output messages created from all, part, or none of the input message (in the Compute node only)

The following topics provide more information about these and other tasks that you can perform with ESQL. Unless otherwise stated, these guidelines apply to messages in all message domains except the BLOB domain, for which you can implement a limited set of actions.

- "Tailoring ESQL code for different node types" on page 326
- "Manipulating message body content" on page 327
- "Manipulating other parts of the message tree" on page 347
- "Transforming from one data type to another" on page 359
- "Adding keywords to ESQL files" on page 366
- "Interaction with databases using ESQL" on page 366
- "Coding ESQL to handle errors" on page 378
- "Accessing broker properties from ESQL" on page 454
- "Configuring a message flow at deployment time with user-defined properties" on page 454

The following topics provide additional information specific to the parser that you have specified for the input message:

- "Manipulating messages in the MRM domain" on page 431
- "Manipulating messages in the XML domain" on page 431
- "Manipulating messages in the XMLNS domain" on page 421
- "Manipulating messages in the XMLNSC domain" on page 408
- "Manipulating messages in the JMS domains" on page 448
- "Manipulating messages in the IDOC domain" on page 448
- "Manipulating messages in the MIME domain" on page 449
- "Manipulating messages in the BLOB domain" on page 451

ESQL examples

Most of the examples included in the topics listed previously show parser-independent ESQL. If examples include a reference to MRM, they assume that you have modeled the message in the MRM and that you have set the names of the MRM objects to be identical to the names of the corresponding tags or attributes in the XML source message. Some examples are also shown for the XML domain. Unless stated otherwise, the principals illustrated are the same for all message domains. For domain-specific information, use the appropriate link in the previous list.

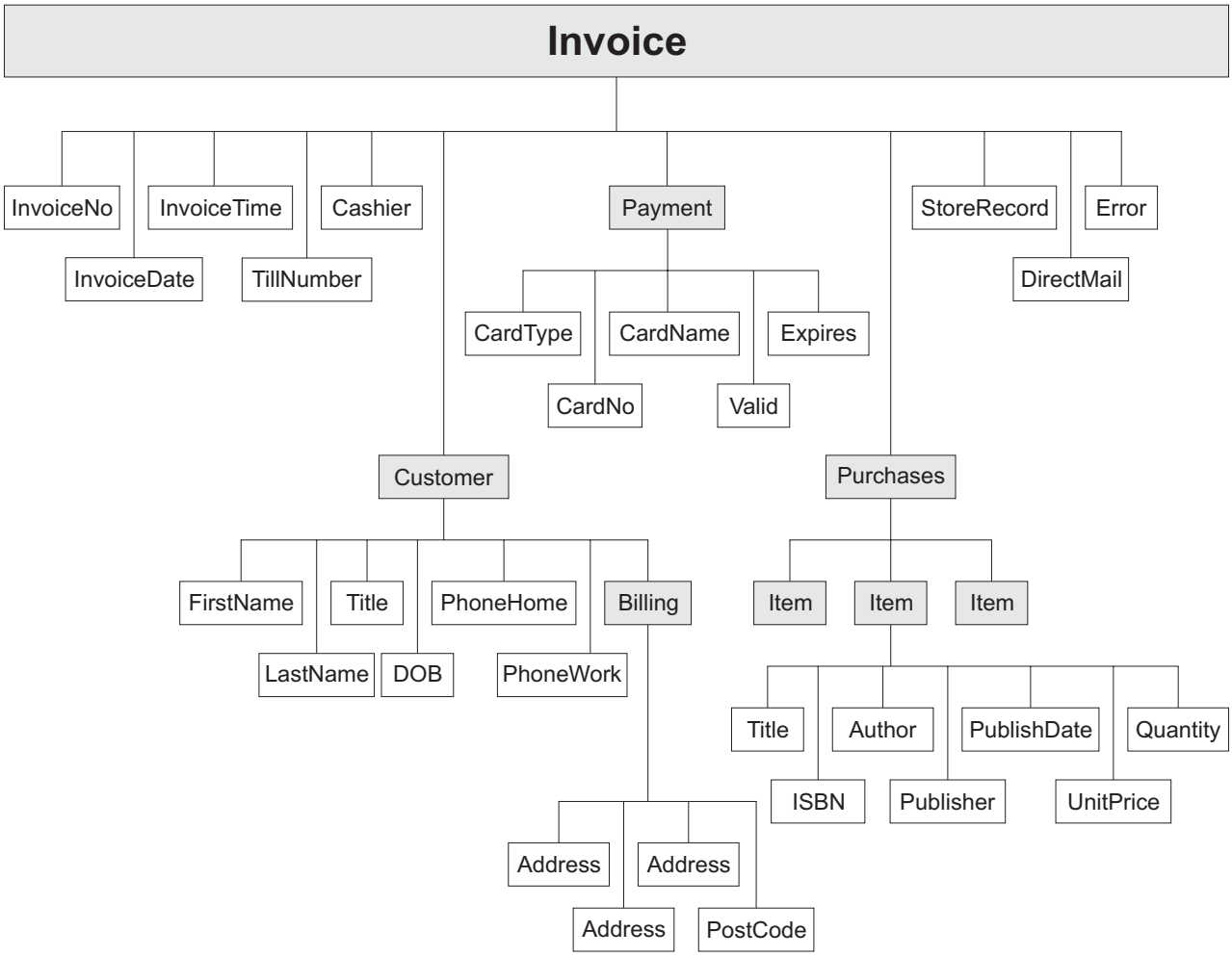
Most of the topics that include example ESQL use the ESQL sample message, Invoice, as the input message to the logic. This message is provided in XML source format (with tags and attributes), see Example message. The example message is shown in the following diagram.

The topics specific to the MRM domain use the message that is created in the following sample:

- Video Rental

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

A few other input messages are used to show ESQL that provides function on messages with a structure or content that is not included in the Invoice or Video samples. Where this occurs, the input message is included in the topic that refers to it.



Tailoring ESQL code for different node types

When you code ESQL to configure Compute, Database, and Filter node behavior, be aware of the limitations of each type of node.

Compute node

You can configure the Compute node to do any of the following operations:

- Update data in a database.
- Insert data into a database.
- Delete data from a database.
- Update the environment tree.
- Update the local environment tree.

- Create one or more output messages, with none, some, or all the content of the input message, and propagate these new messages to the next node in the message flow.

  To propagate the input LocalEnvironment to the output LocalEnvironment, remember to set the Compute node property Compute mode to an appropriate value. The Environment is always propagated in the output message.

Database node

You can configure the Database node to do any of the following operations:

- Update data in a database.
- Insert data into a database.
- Delete data from a database.
- Update the environment tree.
- Update the local environment tree.
- Propagate the input message to the next node in the message flow.

Filter node

You can configure the Filter node to do any of the following operations:

- Update data in a database.
- Insert data into a database.
- Delete data from a database.
- Update the environment tree.
- Update the local environment tree.
- Propagate the input message to the next node in the message flow (the terminal through which the message is propagated depends on the result of the filter expression).

View the remaining tasks in this section to find the details of how you can perform these operations.

Manipulating message body content

The message body is always the last child of root, and is identified by its parser name, for example XML or MRM. You can refer to, modify, and create message body data.

The following topics describe how you can refer to, modify, and create message body data. The information provided here is domain independent.

- "Manipulating repeating fields in a message tree" on page 346

Referencing field types:

Some message parsers have complex models in which it is not enough to identify a field simply by its name and an array subscript. In these cases, you associate an optional field type with an element of data in the tree format.

Each element within the parsed tree can be one of three types:

Name element
A name element has a string, which is the name of the element, associated with it. An example of a name element is `XMLElement`, described in "XML element" on page 1572.

Value element
A value element has a value associated with it. An example of a value element is `XMLContent`, described in "XML content" on page 1572.

Name-value element
A name-value element is an optimization of the case where a name element contains only a value element and nothing else. The element contains both a name and a value. An example of a name-value element is `XMLAttribute`, described in "XML attribute" on page 1570.

Accessing elements in the message body:

When you want to access the contents of a message, for reading or writing, use the structure and arrangement of the elements in the tree that is created by the parser from the input bit stream.

Follow the relevant parent and child relationships from the top of the tree downwards, until you reach the required element.

- If you are referring to the input message tree to interrogate its content in a Compute node, use correlation name `InputBody` followed by the path to the element to which you are referring. `InputBody` is equivalent to `InputRoot` followed by the parser name (for example, `InputRoot.MRM`), which you can use if you prefer.
- If you are referring to the output message tree to set or modify its content in the Compute node, use correlation name `OutputRoot` followed by the parser name (for example, `OutputRoot.MRM`).
- If you are referring to the input message to interrogate its contents in a Database or Filter node, use correlation name `Body` to refer to the start of the message. `Body` is equivalent to `Root` followed by the parser name (for example, `Root.XMLNS`), which you can use if you prefer.

  You must use these different correlation names because there is only one message to which to refer in a Database or Filter node; you cannot create an output message in these nodes. Use a Compute node to create an output message.

When you construct field references, the names that you use must be valid ESQL identifiers that conform to ESQL rules. If you enclose an item in double quotation marks, ESQL interprets it as an identifier. If you enclose an item in single quotation marks, ESQL interprets it as a character literal. You must enclose all strings (character strings, byte strings, or binary (bit) strings) in quotation marks, as shown in the following examples. To include a single or double quotation mark within a string, include two consecutive single or double quotation marks.

Important: For a full description of field reference syntax, see ESQL field reference overview.

For more information about ESQL data types, see ESQL data types in message flows.

Assume that you have created a message flow that handles the message Invoice, shown in the figure in "Writing ESQL" on page 324. If, for example, you want to interrogate the element `CardType` from within a Compute node, use the following statement:

```
IF InputBody.Invoice.Payment.CardType='Visa' THEN
   DO;
     -- more ESQL --
END IF;
```

If you want to make the same test in a Database or Filter node (where the reference is to the single input message), code:

```
IF Body.Invoice.Payment.CardType='Visa' THEN
   DO;
     -- more ESQL --
END IF;
```

If you want to copy an element from an input XML message to an output message in the Compute node without changing it, use the following ESQL:

```
SET OutputRoot.XMLNS.Invoice.Customer.FirstName =
              InputBody.Invoice.Customer.FirstName;
```

If you want to copy an element from an input XML message to an output message and update it, for example by folding to uppercase or by calculating a new value, code:

```
SET OutputRoot.XMLNS.Invoice.Customer.FirstName =
              UPPER(InputBody.Invoice.Customer.FirstName);
SET OutputRoot.XMLNS.Invoice.InvoiceNo = InputBody.Invoice.InvoiceNo + 1000;
```

If you want to set a STRING element to a constant value, code:

```
SET OutputRoot.XMLNS.Invoice.Customer.Title = 'Mr';
```

You can also use the equivalent statement:

```
SET OutputRoot.XMLNS.Invoice.Customer.Title VALUE = 'Mr';
```

If you want to update an INTEGER or DECIMAL, for example the element `TillNumber`, with the value `26`, use the following assignment (valid in the Compute node only):

```
SET OutputRoot.MRM.Invoice.TillNumber=26;
```

The integer data type stores numbers using the 64-bit twos complement form, allowing numbers in the range -9223372036854775808 to 9223372036854775807. You can specify hexadecimal notation for integers as well as normal integer literal format. The hexadecimal letters A to F can be written in uppercase or lowercase, as can the X after the initial zero, which is required. The following example produces the same result as the example shown earlier:

```
SET OutputRoot.MRM.Invoice.TillNumber= 0x1A;
```

The following examples show SET statements for element types that do not appear in the example Invoice message.

To set a FLOAT element to a non-integer value, code:

```
SET OutputRoot.MRM.FloatElement1 = 1.2345e2;
```

To set a BINARY element to a constant value, code:

```
SET OutputRoot.MRM.BinaryElement1 = X'F1F1';
```

For BINARY values, you must use an initial character X (uppercase or lowercase) and enclose the hexadecimal characters (also uppercase or lowercase) in single quotation marks, as shown.

To set a BOOLEAN element to a constant value (the value 1 equates to true, 0 equates to false), code:

```
SET OutputRoot.MRM.BooleanElement1 = true;
```

or

```
SET OutputRoot.MRM.BooleanElement1 = 1;
```

You can use the SELECT statement to filter records from an input message without reformatting the records, and without any knowledge of the complete format of each record. Consider the following example:

```
-- Declare local variable
DECLARE CurrentCustomer CHAR 'Smith';

-- Loop through the input message
SET OutputRoot.XMLNS.Invoice[] =
    (SELECT I FROM InputRoot.XMLNS.Invoice[] AS I
              WHERE I.Customer.LastName = CurrentCustomer
    );
```

This code writes all records from the input Invoice message to the output message if the WHERE condition (LastName = Smith) is met. All records that do not meet the condition are not copied from input message to output message. I is used as an alias for the correlation name InputRoot.XMLNS.Invoice[].

The declared variable CurrentCustomer is initialized on the DECLARE statement: this option is the most efficient way of declaring a variable for which the initial value is known.

You can use this alias technique with other SELECT constructs. For example, if you want to select all the records of the input Invoice message, and create an additional record:

```
-- Loop through the input message
SET OutputRoot.XMLNS.Invoice[] =
    (SELECT I, 'Customer' || I.Customer.LastName AS ExtraField
              FROM InputRoot.XMLNS.Invoice[] AS I
    );
```

You could also include an AS clause to place records in a subfolder in the message tree:

```
-- Loop through the input message
SET OutputRoot.XMLNS.Invoice[] =
    (SELECT I AS Order
            FROM InputRoot.XMLNS.Invoice[] AS I
    );
```

If you are querying or setting elements that contain, or might contain, null values, be aware of the following considerations:

Querying null values

When you compare an element to the ESQL keyword NULL, this tests whether the element is present in the logical tree that has been created from the input message by the parser.

For example, you can check if an invoice number is included in the current Invoice message with the following statement:

```
IF InputRoot.XMLNS.Invoice.InvoiceNo IS NULL THEN
   DO;
     -- more ESQL --
END IF;
```

You can also use an ESQL reference, as shown in the following example:

```
DECLARE cursor REFERENCE TO InputRoot.MRM.InvoiceNo;

IF LASTMOVE(cursor) = FALSE THEN
   SET OutputRoot.MRM.Analysis = 'InvoiceNo does not exist in logical tree';
ELSEIF FIELDVALUE(cursor) IS NULL THEN
   SET OutputRoot.MRM.Analysis =
       'InvoiceNo does exist in logical tree but is defined as an MRM NULL value';
ELSE
   SET OutputRoot.MRM.Analysis = 'InvoiceNo does exist and has a value';
END IF;
```

For more information about declaring and using references, see “Creating dynamic field references” on page 335. For a description of the LASTMOVE and FIELDVALUE functions, see LASTMOVE function and FIELDTYPE function.

If the message is in the MRM domain, there are additional considerations for querying null elements that depend on the physical format. For further details, see “Querying null values in a message in the MRM domain” on page 441.

Setting null values

You can use two statements to set null values:

1. If you set the element to NULL by using the following statement, the element is deleted from the message tree:

```
SET OutputRoot.XMLNS.Invoice.Customer.Title = NULL;
```

   If the message is in the MRM domain, there are additional considerations for null values that depend on the physical format. For further details, see “Setting null values in a message in the MRM domain” on page 441.

   This technique is called implicit null processing.

2. If you set the value of this element to NULL as follows:

```
SET OutputRoot.XMLNS.Invoice.Customer.Title VALUE = NULL;
```

the element is not deleted from the message tree. Instead, a special value of NULL is assigned to the element.

```
SET OutputRoot.XMLNS.Invoice.Customer.Title = NULL;
```

If the message is in the MRM domain, the content of the output bit stream depends on the settings of the physical format null handling properties. For further details, see "Setting null values in a message in the MRM domain" on page 441.

This technique is called explicit null processing.

If you set an MRM complex element or an XML, XMLNS, or JMS parent element to NULL without using the VALUE keyword, that element and all its children are deleted from the logical tree.

Accessing known multiple occurrences of an element:

When you refer to or create the content of messages, it is very likely that the data contains repeating fields. If you know how many instances there are of a repeating field, and you want to access a specific instance of such a field, you can use an array index as part of a field reference.

For example, you might want to filter on the first line of an address, to expedite the delivery of an order. Three instances of the element Billling.Address are always present in the sample message. To test the first line, write an expression such as:

```
IF Body.Invoice.Customer.Billing.Address[1] = 'Patent Office' THEN
   DO;
     -- more ESQL --
END IF;
```

The array index [1] indicates that it is the first instance of the repeating field that you are interested in (array indices start at 1). An array index such as this can be used at any point in a field reference, so you could, for example, filter on the following test:

```
IF Body.Invoice."Item"[1].Quantity> 2 THEN
   DO;
     -- more ESQL --
END IF;
```

You can refer to the last instance of a repeating field using the special [<] array index, and to instances relative to the last (for example, the second to last) as follows:

- `Field[<]` indicates the last element.
- `Field[<1]` indicates the last element.
- `Field[<2]` indicates the last but one element (the penultimate element).

You can also use the array index [>] to represent the first element, and elements relative to the first element in a similar way.

- `Field[>]` indicates the first element. This is equivalent to `Field[1]`.

The following examples refer to the Invoice message using these indexes:

```
IF Body.Invoice.Customer.Billing.Address[<] = 'Hampshire' THEN
      DO;
     -- more ESQL --
END IF;
IF Body.Invoice.Customer.Billing.Address[<2 ] = 'Southampton' THEN
   DO;
     -- more ESQL --
END IF;
```

You can also use these special indexes for elements that repeat an unknown number of times.

Deleting repeating fields:

If you pass a message with several repeats of an element through a message flow and you want to delete some of the repeats, be aware that the numbering of the repeats is reordered after each delete. For example, if you have a message with five repeats of a particular element, and in the message flow you have the following ESQL:

```
SET OutputRoot.MRM.e_PersonName[1] = NULL;
SET OutputRoot.MRM.e_PersonName[4] = NULL;
```

You might expect elements one and four to be deleted. However, because repeating elements are stored on a stack, when you delete one, the one above it takes its place. This means that, in the above example, elements one and five are deleted. To avoid this problem, delete in reverse order, that is, delete element four first, then delete element one.

Accessing unknown multiple occurrences of an element:

To access repeating fields in a message, you must use a construct that can iterate over all instances of a repeating field.

You are likely to deal with messages that contain repeating fields with an unknown number of repeats (such as the `Item` field in Example message).

To write a filter that takes into account all instances of the `Item` field, you need to use a construct that can iterate over all instances of a repeating field. The quantified predicate allows you to execute a predicate against all instances of a repeating field, and collate the results.

For example, you might want to verify that none of the items that are being ordered has a quantity greater than 50. To do this you could write:

```
FOR ALL Body.Invoice.Purchases."Item"[]
    AS I (I.Quantity <= 50)
```

With the quantified predicate, the first thing to note is the brackets [] on the end of the field reference after FOR ALL. These tell you that you are iterating over all instances of the `Item` field.

In some cases, this syntax appears unnecessary because you can get that information from the context, but it is done for consistency with other pieces of syntax.

The AS clause associates the name `I` with the current instance of the repeating field. This is similar to the concept of iterator classes used in some object oriented

languages such as C++. The expression in parentheses is a predicate that is evaluated for each instance of the `Item` field.

A description of this example is:

Iterate over all instances of the field Item inside `Body.Invoice`. For each iteration:

1. Bind the name `I` to the current instance of `Item`.
2. Evaluate the predicate `I.Quantity <= 50`. If the predicate:
   - Evaluates to TRUE for all instances of `Item`, return TRUE.
   - Is FALSE for any instance of `Item`, return FALSE.
   - For a mixture of TRUE and UNKNOWN, return UNKNOWN.

The above is a description of how the predicate is evaluated if you use the ALL keyword. An alternative is to specify SOME, or ANY, which are equivalent. In this case the quantified predicate returns TRUE if the sub-predicate returns TRUE for any instance of the repeating field. Only if the sub-predicate returns FALSE for all instances of the repeating field does the quantified predicate return FALSE. If a mixture of FALSE and UNKNOWN values are returned from the sub-predicate, an overall value of UNKNOWN is returned.

In the following filter expression:

```
FOR ANY Body.Invoice.Purchases."Item"[]
    AS I (I.Title = 'The XML Companion')
```

the sub-predicate evaluates to TRUE. However this next expression returns FALSE:

```
FOR ANY Body.Invoice.Purchases."Item"[]
    AS I (I.Title = 'C Primer')
```

because the `C Primer` is not included on this invoice. If some of the items in the invoice do not include a book title field, the sub-predicate returns UNKNOWN, and the quantified predicate returns the value UNKNOWN.

To deal with the possibility of null values appearing, write this filter with an explicit check on the existence of the field, as follows:

```
FOR ANY Body.Invoice.Purchases."Item"[]
  AS I (I.Book IS NOT NULL AND I.Book.Title = 'C Primer')
```

The predicate IS NOT NULL ensures that, if an `Item` field does not contain a `Book`, a FALSE value is returned from the sub-predicate.

You can also manipulate arbitrary repeats of fields within a message by using a SELECT expression, as described in “Referencing columns in a database” on page 367.

You can refer to the first and last instances of a repeating field using the [>] and [<] array indexes, and to instances relative to the first and last, even if you do not know how many instances there are. These indexes are described in “Accessing known multiple occurrences of an element” on page 332.

Alternatively, you can use the CARDINALITY function to determine how many instances of a repeating field there are. For example:

```
DECLARE I INTEGER CARDINALITY(Body.Invoice.Purchases."Item"[])
```

Using anonymous field references:

You can refer to the array of all children of a particular element by using a path element of `*`.

For example:

```
InputRoot.*[]
```

is a path that identifies the array of all children of InputRoot. This is often used in conjunction with an array subscript to refer to a particular child of an entity by position, rather than by name. For example:

InputRoot.*[<]
: Refers to the last child of the root of the input message, that is, the body of the message.

InputRoot.*[1]
: Refers to the first child of the root of the input message, the message properties.

You might want to find out the name of an element that has been identified with a path of this kind. To do this, use the FIELDNAME function, which is described in FIELDNAME function.

Creating dynamic field references:

You can use a variable of type REFERENCE as a dynamic reference to navigate a message tree. This acts in a similar way to a message cursor or a variable pointer.

It is generally simpler and more efficient to use reference variables in preference to array indexes when you access repeating structures. Reference variables are accepted everywhere. Field references are accepted and come with a set of statements and functions to allow detailed manipulation of message trees.

You must declare a dynamic reference before you can use it. A dynamic reference is declared and initialized in a single statement. The following example shows how to create and use a reference.

```
-- Declare the dynamic reference
DECLARE myref REFERENCE TO OutputRoot.XMLNS.Invoice.Purchases.Item[1];

--  Continue processing for each item in the array
WHILE LASTMOVE(myref)=TRUE
DO
-- Add 1 to each item in the array
   SET myref = myref + 1;
-- Move the dynamic reference to the next item in the array
   MOVE myref NEXTSIBLING;
END WHILE;
```

This example declares a dynamic reference, `myref`, which points to the first item in the array within Purchases. The value in the first item is incremented by one, and the pointer (dynamic reference) is moved to the next item. Once again the item value is incremented by one. This process continues until the pointer moves outside the scope of the message array (all the items in this array have been processed) and the LASTMOVE function returns FALSE.

The following examples show further examples.

```
DECLARE ref1 REFERENCE TO InputBody.Invoice.Purchases.Item[1];

DECLARE ref2 REFERENCE TO
 InputBody.Invoice.Purchases.NonExistentField;

DECLARE scalar1 CHARACTER;
DECLARE ref3 REFERENCE TO scalar1;
```

In the second example, `ref2` is set to point to InputBody because the specified field does not exist.

With the exception of the MOVE statement, which changes the position of the dynamic reference, you can use a dynamic reference anywhere that you can use a static reference. The value of the dynamic reference in any expression or statement is the value of the field or variable to which it currently points. For example, using the message in Example message, the value of `Invoice.Customer.FirstName` is Andrew. If the dynamic reference `myref` is set to point at the FirstName field as follows:

```
DECLARE myref REFERENCE TO Invoice.Customer;
```

the value of `myref` is Andrew. You can extend this dynamic reference as follows:

```
SET myref.Billing.Address[1] = 'Oaklands';
```

This changes the address in the example to Oaklands Hursley Village Hampshire SO213JR.

The position of a dynamic reference remains fixed even if a tree is modified. To illustrate this point the steps that follow use the message in Example message as their input message and create a modified version of this message as an output message:

1. Copy the input message to the output message.
2. To modify the output message, first declare a dynamic reference `ref1` that points at the first item, The XML Companion.

```
DECLARE ref1 REFERENCE TO
 OutputRoot.XMLNS.Invoice.Purchases.Item[1];
```

   The dynamic reference is now equivalent to the static reference `OutputRoot.XMLNS.Invoice.Purchases.Item[1]`.

3. Use a create statement to insert a new first item for this purchase.

```
CREATE PREVIOUSSIBLING OF ref1 VALUES 'Item';
```

   The dynamic reference is now equivalent to the static reference `OutputRoot.XMLNS.Invoice.Purchases.Item[2]`.

Creating new fields:

You can use a Compute node to create a new output message by adding new fields to an existing input message.

This topic provides example ESQL code for a Compute node that creates a new output message based on the input message, to which are added a number of additional fields.

The input message received by the Compute node within the message flow is an XML message, and has the following content:

```
<TestCase description="This is my TestCase">
    <Identifier>ES03B305_T1</Identifier>
    <Sport>Football</Sport>
    <Date>01/02/2000</Date>
    <Type>LEAGUE</Type>
</TestCase>
```

The Compute node is configured and an ESQL module is created that includes the following ESQL. The following code copies the headers from the input message to the new output message, then creates the entire content of the output message body.

```
-- copy headers
DECLARE i INTEGER 1;
DECLARE numHeaders INTEGER CARDINALITY(InputRoot.*[]);

WHILE i < numHeaders DO
    SET OutputRoot.*[i] = InputRoot.*[i];
    SET i = i + 1;
END WHILE;

CREATE FIELD OutputRoot.XMLNS.TestCase.description TYPE NameValue VALUE 'This is my TestCase';
CREATE FIRSTCHILD OF OutputRoot.XMLNS.TestCase Domain('XMLNS') NAME 'Identifier'
                               VALUE InputRoot.XMLNS.TestCase.Identifier;
CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase Domain('XMLNS') NAME 'Sport'
                               VALUE InputRoot.XMLNS.TestCase.Sport;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase Domain('XMLNS') NAME 'Date'
                               VALUE InputRoot.XMLNS.TestCase.Date;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase Domain('XMLNS') NAME 'Type'
                               VALUE InputRoot.XMLNS.TestCase.Type;
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[1].Number TYPE NameValue
                               VALUE 'Premiership';
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[1].Result[1].Number TYPE NameValue VALUE  '1';
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[1].Result[1].Home TYPE Name;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[1].Result[1].Home NAME 'Team'
                               VALUE 'Liverpool' ;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[1].Result[1].Home NAME 'Score'
                               VALUE '4';
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[1].Result[1].Away TYPE Name;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[1].Result[1].Away NAME 'Team'
                               VALUE 'Everton';
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[1].Result[1].Away NAME 'Score'
                               VALUE '0';

 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[1].Result[2].Number TYPE NameValue VALUE '2';
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[1].Result[2].Home TYPE Name;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[1].Result[2].Home NAME 'Team'
                               VALUE 'Manchester United';
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[1].Result[2].Home NAME 'Score'
                               VALUE '2';
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[1].Result[2].Away TYPE Name;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[1].Result[2].Away NAME 'Team'
                               VALUE 'Arsenal';
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[1].Result[2].Away NAME 'Score'
                               VALUE '3';

 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[2].Number TYPE NameValue
                               VALUE '2';
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[2].Result[1].Number TYPE NameValue
```

```
                                VALUE  '1';
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[2].Result[1].Home TYPE Name;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[2].Result[1].Home NAME 'Team'
                                VALUE 'Port Vale';
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[2].Result[1].Home NAME 'Score'
                                VALUE '9' ;
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[2].Result[1].Away TYPE Name;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[2].Result[1].Away NAME 'Team'
                                VALUE 'Brentford';
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[2].Result[1].Away NAME 'Score'
                                VALUE '5';
```

The output message that results from the ESQL shown above has the following structure and content:

```
<TestCase description="This is my TestCase">
  <Identifier>ES03B305_T1</Identifier>
  <Sport>Football</Sport>
  <Date>01/02/2000</Date>
  <Type>LEAGUE</Type>
  <Division Number="Premiership">
    <Result Number="1">
      <Home>
        <Team>Liverpool</Team>
        <Score>4</Score>
      </Home>
      <Away>
        <Team>Everton</Team>
        <Score>0</Score>
      </Away>
    </Result>
    <Result Number="2">
      <Home>
        <Team>Manchester United</Team>
        <Score>2</Score>
      </Home>
      <Away>
        <Team>Arsenal</Team>
        <Score>3</Score>
      </Away>
    </Result>
  </Division>
  <Division Number="2">
    <Result Number="1">
      <Home>
        <Team>Port Vale</Team>
        <Score>9</Score>
      </Home>
      <Away>
        <Team>Brentford</Team>
        <Score>5</Score>
      </Away>
    </Result>
  </Division>
</TestCase>
```

Generating multiple output messages:

You can use the PROPAGATE statement to generate multiple output messages in the Compute node. The output messages that you generate can have the same or different content. You can also direct output messages to any of the four alternate output terminals of the Compute node, or to a Label node.

For example, to create three copies of the input message received by the Compute node, and send one to the standard Out terminal of the Compute node, one to the

first alternate Out1 terminal of the Compute node, and one to the Label node ThirdCopy, code the following ESQL:

```
SET OutputRoot = InputRoot;
PROPAGATE;
SET OutputRoot = InputRoot;
PROPAGATE TO TERMINAL 'out1';
SET OutputRoot = InputRoot;
PROPAGATE TO LABEL 'ThirdCopy';
```

In the above example, the content of OutputRoot is reset before each PROPAGATE, because by default the node clears the output message buffer and reclaims the memory when the PROPAGATE statement completes. An alternative method is to instruct the node not to clear the output message on the first two PROPAGATE statements, so that the message is available for routing to the next destination. The code to achieve this result is:

```
SET OutputRoot = InputRoot;
PROPAGATE DELETE NONE;
SET OutputRoot = InputRoot;
PROPAGATE TO TERMINAL 'out1' DELETE NONE;
SET OutputRoot = InputRoot;
PROPAGATE TO LABEL 'ThirdCopy';
```

If you do not initialize the output buffer, an empty message is generated, and the message flow detects an error and throws an exception.

Also ensure that you copy all required message headers to the output message buffer for each output message that you propagate.

If you want to modify the output message content before propagating each message, code the appropriate ESQL to make the changes that you want before you code the PROPAGATE statement.

If you set up the contents of the last output message that you want to generate, and propagate it as the final action of the Compute node, you do not have to include the final PROPAGATE statement. The default action of the Compute node is to propagate the contents of the output buffer when it terminates. This is implemented by the RETURN TRUE statement, included as the final statement in the module skeleton.

For example, to generate three copies of the input message, and not perform any further action, include this code immediately before the RETURN TRUE statement:

```
SET OutputRoot = InputRoot;
PROPAGATE DELETE NONE;
PROPAGATE DELETE NONE;
```

Alternatively, you can modify the default behavior of the node by changing RETURN TRUE to RETURN FALSE:

```
SET OutputRoot = InputRoot;
PROPAGATE DELETE NONE;
PROPAGATE DELETE NONE;
PROPAGATE;
RETURN FALSE;
```

Three output messages are generated by the three PROPAGATE statements. The final RETURN FALSE statement causes the node to terminate but not propagate a

final output message. Note that the final PROPAGATE statement does not include the DELETE NONE clause, because the node must release the memory at this stage.

Using numeric operators with datetime values:

The following examples show the ESQL that you can code to manipulate datetime values with numeric operators.

Adding an interval to a datetime value

The simplest operation that you can perform is to add an interval to, or subtract an interval from, a datetime value. For example, you could write the following expressions:

```
DATE '2000-03-29' + INTERVAL '1' MONTH
TIMESTAMP '1999-12-31 23:59:59' + INTERVAL '1' SECOND
```

The following example shows how to calculate a retirement date by adding the retirement age to the birth date.

```
DECLARE retAge CHARACTER '65';
DECLARE birthDate DATE DATE '1953-06-01';

SET OutputRoot.XML.Test.retirementDate = birthDate + CAST(retAge AS INTERVAL YEAR);
```

The repetition of the word DATE in the above example is intentional. The first occurrence of DATE specifies the data type of the declared variable, birthDate. The second occurrence initializes the same variable with the constant character string that is enclosed in single quotation marks as a DATE.

Adding or subtracting two intervals

You can combine two interval values by using addition or subtraction. The two interval values must be of compatible types. It is not valid to add a year-month interval to a day-second interval as shown in the following example:

```
INTERVAL '1-06' YEAR TO MONTH + INTERVAL '20' DAY
```

The interval qualifier of the resultant interval is sufficient to encompass all the fields that are present in the two operand intervals. For example:

```
INTERVAL '2 01' DAY TO HOUR + INTERVAL '123:59' MINUTE TO SECOND
```

results in an interval with qualifier DAY TO SECOND, because both day and second fields are present in at least one of the operand values.

Subtracting two datetime values

You can subtract two datetime values to return an interval. You must include an interval qualifier in the expression to indicate what precision the result should be returned in. For example:

```
(CURRENT_DATE - DATE '1776-07-04') DAY
```

returns the number of days since the 4th July 1776, whereas:

```
(CURRENT_TIME - TIME '00:00:00') MINUTE TO SECOND
```

returns the age of the day in minutes and seconds.

Scaling intervals

You can multiply or divide an interval value by an integer factor:

```
INTERVAL '2:30' MINUTE TO SECOND / 4
```

Calculating a time interval:

You can use ESQL to calculate the time interval between two events, and to set a timer to be triggered after a specified interval.

This ESQL example calculates the time interval between an input WebSphere MQ message being put on the input queue, and the time that it is processed in the current Compute node.

(When you make a call to a CURRENT_ datetime function, the value that is returned is identical to the value returned to another call in the same node. This ensures that you can use the function consistently within a single node.)

```
 CALL CopyMessageHeaders();
 Declare PutTime INTERVAL;

 SET PutTime = (CURRENT_GMTTIME - InputRoot.MQMD.PutTime) MINUTE TO SECOND;

 SET OutputRoot.XMLNS.Test.PutTime = PutTime;
```

The output message has the format (although actual values vary):

```
<Test>
 <PutTime>INTERVAL &apos;1:21.862&apos; MINUTE TO SECOND</PutTime>
</Test>
```

The following code snippet sets a timer, to be triggered after a specified interval from the start of processing, in order to check that processing has completed. If processing has not completed within the elapsed time, the firing of the timer might, for example, trigger some recovery processing.

The `StartTime` field of the timeout request message is set to the current time plus the allowed delay period, which is defined by a user-defined property on the flow. (The user-defined property has been set to a string of the form "HH:MM:SS" by the administrator.)

```
DECLARE StartDelyIntervalStr EXTERNAL CHARACTER '01:15:05';

 CREATE PROCEDURE ValidateTimeoutRequest() BEGIN

  -- Set the timeout period
  DECLARE timeoutStartTimeRef REFERENCE TO
          OutputRoot.XMLNSC.Envelope.Header.TimeoutRequest.StartTime;
  IF LASTMOVE(timeoutStartTimeRef)
   THEN
   -- Already set
  ELSE
   -- Set it from the UDP StartDelyIntervalStr
   DECLARE startAtTime TIME CURRENT_TIME
          + CAST(StartDelyIntervalStr AS INTERVAL HOUR TO SECOND);

   -- Convert "TIME 'hh.mm.ss.fff'" to hh.mm.ss format
       -- needed in StartTime field
   DECLARE startAtTimeStr CHAR;
   SET startAtTimeStr = startAtTime;
   SET startAtTimeStr = SUBSTRING(startAtTimeStr FROM 7 FOR  8);
```

```
   SET OutputRoot.XMLNSC.Envelope.Header.TimeoutRequest.StartTime
                            = startAtTimeStr;
  END IF;
 END;
```

Selecting a subfield from a larger field:

You might have a message flow that processes a message containing delimited subfields. You can code ESQL to extract a subfield from the surrounding content if you know the delimiters of the subfield.

If you create a function that performs this task, or a similar one, you can invoke it both from ESQL modules (for Compute, Database, and Filter nodes) and from mapping files (used by DataDelete, DataInsert, DataUpdate, Extract, Mapping, and Warehouse nodes).

The following function example extracts a particular subfield of a message that is delimited by a specific character.

```
CREATE FUNCTION SelectSubField
       (SourceString CHAR, Delimiter CHAR, TargetStringPosition INT)
       RETURNS CHAR
-- This function returns a substring at parameter position TargetStringPosition within the
-- passed parameter SourceString.  An example of use might be:
-- SelectSubField(MySourceField,' ',2) which will select the second subfield from the
-- field MySourceField delimited by a blank.  If MySourceField has the value
-- "First Second Third" the function will return the value "Second"
  BEGIN
    DECLARE DelimiterPosition INT;
    DECLARE CurrentFieldPosition INT 1;
    DECLARE StartNewString INT 1;
    DECLARE WorkingSource CHAR SourceString;
    SET DelimiterPosition = POSITION(Delimiter IN SourceString);
    WHILE CurrentFieldPosition < TargetStringPosition
      DO
       IF DelimiterPosition = 0  THEN
      -- DelimiterPosition will be 0 if the delimiter is not found
         -- exit the loop
        SET CurrentFieldPosition = TargetStringPosition;
       ELSE
        SET StartNewString = DelimiterPosition + 1;
        SET WorkingSource = SUBSTRING(WorkingSource FROM StartNewString);
        SET DelimiterPosition = POSITION(Delimiter IN WorkingSource);
        SET CurrentFieldPosition = CurrentFieldPosition + 1;
       END IF;
    END WHILE;
    IF DelimiterPosition> 0 THEN
       -- Remove anything following the delimiter from the string
       SET WorkingSource = SUBSTRING(WorkingSource FROM 1 FOR DelimiterPosition);
       SET WorkingSource = TRIM(TRAILING Delimiter FROM WorkingSource);
    END  IF;
    RETURN WorkingSource;
END;
```

Copying repeating fields:

You can configure a node with ESQL to copy repeating fields in several ways.

Consider an input XML message that contains a repeating structure:

```
...
 <Field_top>
   <field1></field1>
   <field1></field1>
   <field1></field1>
```

```
   <field1></field1>
   <field1></field1>
 </Field_top>
.....
```

You cannot copy this whole structure field with the following statement:

```
SET OutputRoot.XMLNS.Output_top.Outfield1 = InputRoot.XMLNS.Field_top.field1;
```

That statement copies only the first repeat, and therefore produces the same result as this statement:

```
SET OutputRoot.XMLNS.Output_top.Outfield1[1] = InputRoot.XMLNS.Field_top.field1[1];
```

You can copy the fields within a loop, controlling the iterations with the CARDINALITY of the input field:

```
SET I = 1;
SET J = CARDINALITY(InputRoot.XMLNS.Field_top.field1[]);
WHILE  I <= J DO
 SET OutputRoot.XMLNS.Output_top.Outfield1[I] = InputRoot.XMLNS.Field_top.field1[I];
 SET I = I + 1;
END WHILE;
```

This might be appropriate if you want to modify each field in the output message as you copy it from the input field (for example, add a number to it, or fold its contents to uppercase), or after it has been copied. If the output message already contained more Field1 fields than existed in the input message, the surplus fields would not be modified by the loop and would remain in the output message.

The following single statement copies the iterations of the input fields to the output fields, and deletes any surplus fields in the output message.

```
SET OutputRoot.XMLNS.Output_top.Outfield1.[] = InputRoot.XMLNS.Field_top.field1[];
```

The following example shows how you can rename the elements when you copy them into the output tree. This statement does not copy across the source element name, therefore each `field1` element becomes a `Target` element.

```
SET OutputRoot.XMLNS.Output_top.Outfield1.Target[] =
    (SELECT I FROM InputRoot.XMLNS.Field_top.field1[] AS I );
```

The next example shows a different way to do the same operation; it produces the same end result.

```
SET OutputRoot.XMLNS.Output_top.Outfield2.Target[]
              = InputRoot.XMLNS.Field_top.field1[];
```

The following example copies across the source element name. Each `field1` element is retained as a `field1` element under the `Target` element.

```
SET OutputRoot.XMLNS.Output_top.Outfield3.Target.[]
              = InputRoot.XMLNS.Field_top.field1[];
```

This example is an alternative way to achieve the same result, with `field1` elements created under the `Target` element.

```
SET OutputRoot.XMLNS.Output_top.Outfield4.Target.*[]
              = InputRoot.XMLNS.Field_top.field1[];
```

These examples show that there are several ways in which you can code ESQL to copy repeating fields from source to target. Select the most appropriate method to achieve the results that you require.

The principals shown here apply equally to all areas of the message tree to which you can write data, not just the output message tree.

A note about copying fields:

When you copy an input message element to an output element, the *value* and *type* of the output element is set to that of the input element. Therefore, if, for example, you have an input XML document with an attribute, and you want to set a Field element (rather than an attribute) in your output message to the value of the input attribute, you must include a TYPE clause cast to change the element-type from attribute to Field.

For example, given the following input:

```
<Field01 Attrib01='Attrib01_Value'>Field01_Value</Field01>
```

To create an output, such as in the following example:

```
<MyField_A MyAttrib_A='Attrib01_Value' MyAttrib_B='Field01_Value' >
  <MyField_B>Field01_Value</MyField_BC>
  <MyField_C>Attrib01_Value</MyField_C>
 </MyField_A'>
```

You would use the following ESQL:

```
-- Create output attribute from input attribute
SET OutputRoot.XMLNSC.MyField_A.MyAttrib_A = InputRoot.XMLNSC.Field01.Attrib01;
-- Create output field from input field
SET OutputRoot.XMLNSC.MyField_A.MyField_B = InputRoot.XMLNSC.Field01;

-- Create output attribute from input field value, noting we have to
-- "cast" back to an attribute  element
SET OutputRoot.XMLNSC.MyField_A.(XMLNSC.Attribute)MyAttrib_B =
               InputRoot.XMLNSC.Field01;

-- Create output field from input attribute value, noting we have to
-- "cast" back to a field element
SET OutputRoot.XMLNSC.MyField_A.(XMLNSC.Field)MyField_C =
               InputRoot.XMLNSC.Field01.Attrib01;
```

Working with elements of type list:

The XML Schema specification allows an element, or attribute, to contain a list of values that are based on a simple type, with the individual values separated by white space.

In the message tree, an `xsd:list` element is represented as a name node, with an anonymous child node for each list item. Repeating lists can be handled without any loss of information.

Consider the following XML input message:

```
 <message1>
   <listE1>one two three</listE1>
 </message1>
```

This XML element produces the following message tree:

```
 MRM
 listEl  (Name)
  "one"   (Value)
  "two"   (Value)
  "three" (Value)
```

Individual list items can be accessed as `ElementName.*[n]`.

For example, use the following ESQL to access the third item of listE1:

```
SET X = FIELDVALUE (InputBody.message1.listE1.*[3]);
```

An attribute can also be of type `xsd:list`.

Consider the following XML input message:

```
<message1>
<Element listAttr="one two three"/>
</message1>
```

This XML element produces the following message tree:

```
MRM
Element (Name)
 listAttr (Name)
  "one" (Value)
  "two" (Value)
  "three" (Value)
```

As before, individual list items can be accessed as `AttrName.*[n]`.

For example, use the following ESQL to access the third item of listAttr:

```
SET X = FIELDVALUE (InputBody.message1.Element.listAttr.*[3]);
```

A list element can occur more than once.

Consider the following XML message:

```
<message1>
<listE1>one two three/listE1>
<listE1>four five six/listE1>
</message1>
```

The message tree for this XML message is:

```
MRM
listE1 (Name)
 "one" (Value)
 "two" (Value)
 "three" (Value)
listE1 (Name)
 "four" (Value)
 "five" (Value)
 "six" (Value)
```

Code the following ESQL to access the first item in the second occurrence of the list:

```
SET X = FIELDVALUE (InputBody.message1.listE1[2].*[1]);
```

Mapping between a list and a repeating element:

Consider the form of the following XML input message:

```
<MRM>
  <inner>abcde fghij 12345</inner>
</MRM>
```

where the element inner is of type `xsd:list`, and therefore has three associated string values, rather than a single value.

To copy the three values into an output message, where each value is associated with an instance of repeating elements as shown here:

```
<MRM>
  <str1>abcde</str1>
  <str1>fghij</str1>
  <str1>12345</str1>
</MRM>
```

you might expect that the following ESQL syntax works:

```
DECLARE D INTEGER;
SET D = CARDINALITY(InputBody.str1.*[]);
DECLARE M INTEGER 1;
WHILE M <= D DO
   SET OutputRoot.MRM.str1[M] = InputBody.inner.*[M];
   SET M = M + 1;
END WHILE;
```

However, the statement:

```
SET OutputRoot.MRM.str1[M] = InputBody.inner.*[M];
```

requests a tree copy from source to target. Because the target element does not yet exist, the statement creates it, and its value and type are set from the source.

Therefore, to create the output message with the required format, given an input element which is of type `xsd:list`, use the FIELDVALUE function to explicitly retrieve only the value of the source element:

```
SET OutputRoot.MRM.str1[M] = FIELDVALUE(InputBody.inner.*[M]);
```

Manipulating repeating fields in a message tree:

This topic describes the use of the SELECT function, and other column functions, to manipulate repeating fields in a message tree.

Suppose that you want to perform a special action on invoices that have a total order value greater than a certain amount. To calculate the total order value of an `Invoice` field, you must multiply the `Price` fields by the `Quantity` fields in all the `Items` in the message, and total the result. You can do this using a SELECT expression as follows:

```
(
 SELECT SUM( CAST(I.Price AS DECIMAL) * CAST(I.Quantity AS INTEGER) )
  FROM Body.Invoice.Purchases."Item"[] AS I
)
```

The example assumes that you need to use CAST expressions to cast the string values of the fields `Price` and `Quantity` into the correct data types. The cast of the `Price` field into a decimal produces a decimal value with the *natural* scale and precision, that is, whatever scale and precision is necessary to represent the number. These CASTs would not be necessary if the data were already in an appropriate data type.

The SELECT expression works in a similar way to the quantified predicate, and in much the same way that a SELECT works in standard database SQL. The FROM clause specifies what is being iterated, in this case, all `Item` fields in `Invoice`, and establishes that the current instance of `Item` can be referred to using `I`. This form of SELECT involves a column function, in this case the SUM function, so the SELECT is evaluated by adding together the results of evaluating the expression inside the SUM function for each `Item` field in the `Invoice`. As with standard SQL, NULL

values are ignored by column functions, with the exception of the COUNT column function explained later in this section, and a NULL value is returned by the column function only if there are no non-NULL values to combine.

The other column functions that are provided are MAX, MIN, and COUNT. The COUNT function has two forms that work in different ways with regard to NULLs. In the first form you use it much like the SUM function above, for example:

```
SELECT COUNT(I.Quantity)
  FROM Body.Invoice.Purchases."Item"[] AS I
```

This expression returns the number of `Item` fields for which the `Quantity` field is non-NULL. That is, the COUNT function counts non-NULL values, in the same way that the SUM function adds non-NULL values. The alternative way of using the COUNT function is as follows:

```
SELECT COUNT(*)
  FROM Body.Invoice.Purchases."Item"[] AS I
```

Using COUNT(*) counts the total number of `Item` fields, regardless of whether any of the fields is NULL. The above example is in fact equivalent to using the CARDINALITY function, as in the following:

```
CARDINALITY(Body.Invoice.Purchases."Item"[]
```

In all the examples of SELECT given here, just as in standard SQL, you could use a WHERE clause to provide filtering on the fields.

Manipulating other parts of the message tree

You can access message tree headers, the properties tree, the local environment tree, the environment tree and the exception list tree.

The following topics describe how you can access parts of the message tree other than the message body data. These parts of the logical tree are independent of the domain in which the message exists, and all these topics apply to messages in the BLOB domain in addition to all other supported domains.

- "Accessing headers"
- "Accessing the Properties tree" on page 351
- "Accessing the local environment tree" on page 353
- "Accessing the environment tree" on page 357
- "Accessing the ExceptionList tree using ESQL" on page 357

Accessing headers:

If the input message received by an input node includes message headers that are recognized by the input node, the node invokes the correct parser for each header. You can access these headers using ESQL.

Parsers are supplied for most WebSphere MQ headers. The following topics provide some guidance for accessing the information in the MQMD, MQRFH2, and MQPCF headers, which you can follow as general guidance for accessing other headers also present in your messages.

Every header has its own correlation name, for example, MQMD, and you must use this name in all ESQL statements that refer to or set the content of this tree:

- "Accessing the MQMD header"
- "Accessing the MQRFH2 header"
- "Accessing the MQCFH header" on page 349

For further details of the contents of these and other WebSphere MQ headers for which WebSphere Message Broker provides a parser, see "Element definitions for message parsers" on page 1517.

Accessing transport headers:

You can manipulate WebSphere MQ, HTTP, and JMS transport headers and their properties without writing Compute nodes:

- Use the MQHeader node to add, modify, or delete MQ Message Descriptor (MQMD) and MQ Dead Letter Header (MQDLH) headers.
- Use the HTTPHeader node to add, modify, or delete HTTP headers such as HTTPRequest and HTTPReply.
- Use the JMSHeader node to modify contents of the JMS Header_Values and Application properties so that you can control the node's output without programming.

Accessing the MQMD header:

Code ESQL statements to access the fields of the MQMD header.

WebSphere MQ, WebSphere MQ Everyplace, and SCADA messages include an MQMD header.

You can refer to the fields within the MQMD, and you can update them in a Compute node.

For example, you might want to copy the message identifier MSGID in the MQMD to another field in your output message. To do that, code:

```
SET OutputRoot.MRM.Identifier = InputRoot.MQMD.MsgId;
```

If you send a message to an EBCDIC system from a distributed system, you might need to convert the message to a compatible CodedCharSetId and Encoding. To do this conversion, code the following ESQL in the Compute node:

```
SET OutputRoot.MQMD.CodedCharSetId = 500;
SET OutputRoot.MQMD.Encoding = 785;
```

The MQMD properties of CodedCharSetId and Encoding define the code page and encoding of the section of the message that follows (typically this is either the MQRFH2 header or the message body itself).

Differences exist in the way the Properties folder and the MQMD folder are treated with respect to which folder takes precedence for the same fields. For more information, see "Properties versus MQMD folder behavior for various transports" on page 73.

Accessing the MQRFH2 header:

Code ESQL statements to access the fields of the MQRFH2 header.

When you construct an MQRFH2 header in a Compute node, it includes two types of fields:

- Fields in the MQRFH2 header structure; for example, Format and NameValueCCSID.
- Fields in the MQRFH2 NameValue buffer; for example, mcd and psc.

To differentiate between these two field types, insert a value in front of the referenced field in the MQRFH2 field to identify its type; a value for the NameValue buffer is not required because this is the default. The value that you specify for the header structure is `(MQRFH2.Field)`.

For example:

- To create or change an MQRFH2 Format field, specify the following ESQL:

```
SET OutputRoot.MQRFH2.(MQRFH2.Field)Format = 'MQSTR   ';
```

- To create or change the psc folder with a topic:

```
SET OutputRoot.MQRFH2.psc.Topic = 'department';
```

- To add an MQRFH2 header to an outgoing message that is to be used to make a subscription request:

```
DECLARE I INTEGER 1;
DECLARE J INTEGER CARDINALITY(InputRoot.*[]);

WHILE I < J DO
 SET OutputRoot.*[I] = InputRoot.*[I];
 SET I=I+1;
END WHILE;

SET OutputRoot.MQRFH2.(MQRFH2.Field)Version = 2;
SET OutputRoot.MQRFH2.(MQRFH2.Field)Format = 'MQSTR';
SET OutputRoot.MQRFH2.(MQRFH2.Field)NameValueCCSID = 1208;
SET OutputRoot.MQRFH2.psc.Command = 'RegSub';
SET OutputRoot.MQRFH2.psc.Topic = "InputRoot"."MRM"."topel";
SET OutputRoot.MQRFH2.psc.QMgrName = 'DebugQM';
SET OutputRoot.MQRFH2.psc.QName = 'PUBOUT';
SET OutputRoot.MQRFH2.psc.RegOpt = 'PersAsPub';
```

  Variable J is initialized to the value of the cardinality of the existing headers in the message. Using a variable is more efficient than calculating the cardinality on each iteration of the loop, which happens if you code the following WHILE statement:

```
WHILE I < CARDINALITY(InputRoot.*[]) DO
```

The MQRFH2 header can be parsed using either the MQRFH2 parser or the MQRFH2C compact parser. To consume less memory, use the MQRFH2C compact parser by selecting the `Use MQRFH2C compact parser for MQRFH2 Header` check box on the input node of the message flow. This results in paths that contain MQRFH2C instead of MQRFH2; for example: `SET OutputRoot.MQRFH2C.psc.Topic = 'department';`

Target MQRFH2 fields are created only if the headers are copied, and the MQRFH2C parser option is not selected on the MQInput node. In all other circumstances, an MQRFH2C field is created on output.

Accessing the MQCFH header:

Code ESQL statements to access the fields of the MQCFH header (root name MQPCF).

Messages that are of PCF format (MQPCF, MQADMIN, and MQEVENT) include the MQCFH header. You can process the contents of the MQCFH header, accessing parameters, parameter lists, strings, and groups.

The *ParameterCount* field is hidden from view to keep the value synchronized with the true number of parameters. As a result, you cannot directly view, access, or edit the *ParameterCount* field and this applies when you are using:

- ESQL
- The Mapping Node
- Trace, or debugging code.

You can implement your own *ParameterCount* field with a specific value, but this value will be overwritten by the actual number of parameters on the flow exit.

- To access or to construct parameters in the header, use the following syntax:

  ```
  SET OutputRoot.MQPCF.Parameter[nn] =
   Integer parameter ID
  ```

  If you access a 64-bit parameter, use the following syntax to differentiate from 32-bit parameters:

  ```
  SET OutputRoot.MQPCF.Parameter64[nn] =
   Integer parameter ID
  ```

  For example:

  ```
  SET OutputRoot.MQPCF.Parameter[1] =
   MQCACF_AUTH_PROFILE_NAME;
  ```

- For parameter lists, use the following syntax:

  ```
  SET OutputRoot.MQPCF.ParameterList64[nn] =
   Integer parameter ID
  SET OutputRoot.MQPCF.ParameterList64[nn].*[xx] =
   Integer parameter values
  ```

  For example:

  ```
  SET OutputRoot.MQPCF.ParameterList[1] =
   MQIACF_AUTH_ADD_AUTHS;
  SET OutputRoot.MQPCF.ParameterList[1].*[1] =
   MQAUTH_SET;
  SET OutputRoot.MQPCF.ParameterList[1].*[2] =
   MQAUTH_SET_ALL_CONTEXT;
  ```

- A byte string is a byte array data type, and is supported with this syntax:

  ```
  SET OutputRoot.MQPCF.Parameter[nn] =
   Integer parameter ID
  SET OutputRoot.MQPCF.Parameter[nn].* =
   Integer, String or ByteArray Parameter value
  ```

- A group is implemented as a folder containing more PCF, and requires the following syntax:

  ```
  SET OutputRoot.MQPCF.Group[xx] =
   Group Parameter ID
  ```

  For example:

```
SET OutputRoot.MQPCF.Group[1] =
 MQGACF_Q_ACCOUNTING_DATA;
SET OutputRoot.MQPCF.Group[1].Parameter[1] =
 MQCA_CREATION_DATE;
SET OutputRoot.MQPCF.Group[1].Parameter[1].* =
 '2007-02-05';
```

You can also use nested groups; for example;

```
SET OutputRoot.MQPCF.Group[1].Group[1] =
 MQGACF_Q_ACCOUNTING_DATA;
SET OutputRoot.MQPCF.Group[1].Group[1].Parameter[1] =
 MQCA_CREATION_DATE;
SET OutputRoot.MQPCF.Group[1].Group[1].Parameter[1].* =
 '2007-02-05';
```

- A filter is almost identical to a parameter, but it also includes an operator. Therefore the syntax is a tree with named values:

```
SET OutputRoot.MQPCF.Filter[xx] =
 Integer parameter ID
SET OutputRoot.MQPCF.Filter[xx].Operator =
 Integer Filter name
SET OutputRoot.MQPCF.Filter[xx].Value =
 Byte, Integer or String Filter Value
```

- The following is sample code that can be used as an example to create an MQPCF message in a Compute node:

```
CREATE NEXTSIBLING OF OutputRoot.Properties DOMAIN 'MQMD';
CREATE NEXTSIBLING OF OutputRoot.MQMD DOMAIN 'MQADMIN'
NAME 'MQPCF';
CREATE FIELD OutputRoot.MQPCF;
SET OutputRoot.MQMD.MsgType = MQMT_REQUEST;
SET OutputRoot.MQMD.ReplyToQ = 'REPLYQ';
DECLARE refRequest REFERENCE TO OutputRoot.MQPCF;
SET refRequest.Type = 16; --MQCFT_COMMAND_XR z/OS
SET refRequest.StrucLength = MQCFH_STRUC_LENGTH;
SET refRequest.Version = 3; -- required for z/OS
SET refRequest.Command = MQCMD_INQUIRE_Q;
SET refRequest.MsgSeqNumber = 1;
SET refRequest.Control = MQCFC_LAST;
/* First parameter: Queue Name. */
SET refRequest.Parameter[1] = MQCA_Q_NAME;
SET refRequest.Parameter[1].* = 'QUEUENAME.*';
SET refRequest.ParameterList[1] = MQIACF_Q_ATTRS;
SET refRequest.ParameterList[1].*  = MQIACF_ALL;
```

Accessing the Properties tree:

The Properties tree has its own correlation name, Properties, and you must use this in all ESQL statements that refer to or set the content of this tree.

The fields in the Properties tree contain values that define the characteristics of the message. For example, the Properties tree contains message template information for model-driven parsers, fields for the encoding and CCSID in which message data is encoded, and fields that hold the security identity of the message. For a full list of fields in this tree, see “Data types for elements in the Properties subtree” on page 1518.

You can interrogate and update these fields using the appropriate ESQL statements. If you create a new output message in the Compute node, you must set values for the message properties.

Setting output message properties:

If you use the Compute node to generate a new output message, you must set its properties in the Properties tree. The output message properties do not have to be the same as the input message properties.

For example, to set the output message properties for an output MRM message, set the following properties:

Property	Value
MessageSet	Message set identifier
MessageType	Message name[1]
MessageFormat	Physical format name[2]

Notes:

1. For details of the syntax of Message type, see Specifying namespaces in the Message Type property.
2. The name that you specify for the physical format must match the name that you have defined for it. The default physical format names are Binary1, XML1, and Text1.

This ESQL procedure sets message properties to values passed in by the calling statement. You might find that you have to perform this task frequently, and you can use a procedure such as this in many different nodes and message flows. If you prefer, you can code ESQL that sets specific values.

```
CREATE PROCEDURE setMessageProperties(IN OutputRoot REFERENCE, IN setName char,
                 IN typeName char, IN formatName char) BEGIN
 /*****************************************************************************
 * A procedure that sets the message properties
 *****************************************************************************/
 set OutputRoot.Properties.MessageSet    = setName;
 set OutputRoot.Properties.MessageType   = typeName;
 set OutputRoot.Properties.MessageFormat = formatName;
END;
```

To set the output message domain, you can code ESQL statements that refer to the required domain in the second qualifier of the SET statement, the parser field. For example, the ESQL statement sets the domain to MRM:

```
SET OutputRoot.MRM.Field1 = 'field1 data';
```

This ESQL statement sets the domain to XMLNS:

```
SET OutputRoot.XMLNS.Field1 = 'field1 data';
```

Do not specify more than one domain in the ESQL for any single message. However, if you use PROPAGATE statements to generate several output messages, you can set a different domain for each message.

For information about the full list of elements in the Properties tree, see “Data types for elements in the Properties subtree” on page 1518.

Differences exist in the way the Properties folder and the MQMD folder are treated with respect to which folder takes precedence for the same fields. For more information, see “Properties versus MQMD folder behavior for various transports” on page 73.

Accessing the local environment tree:

The local environment tree has its own correlation name, LocalEnvironment, and you must use this name in all ESQL statements that refer to or set the content of this tree.

The local environment tree is used by the broker, and you can refer to and modify this information. You can also extend the tree to contain information that you create yourself. You can create subtrees of this tree that you can use as a scratchpad or working area.

The message flow sets up information in two subtrees, Destination and WrittenDestination, below the LocalEnvironment root. You can refer to the content of both of these subtrees, and you can write to the Destination tree to influence the way in which the message flow processes your message. However, if you write to the Destination tree, follow the defined structure to ensure that the tree remains valid.

The WrittenDestination subtree contains the addresses to which the message has been written. Its name is fixed and it is created by the message flow when a message is propagated through the Out terminal of a request, output, or reply node. The subtree includes transport-specific information (for example, if the output message has been put to a WebSphere MQ queue, it includes the queue manager and queue names). You can use one of the following methods to obtain information about the details of a message after it has been sent by the nodes:

- Connect a Compute node to the Out terminal.
- Configure a user exit to process an output message callback event, as described in "Exploiting user exits" on page 239.

The topic for each node that supports WrittenDestination information contains details about the data that it contains.

If you want the local environment tree to be included in the output message that is propagated by the Compute node, you must set the Compute node property Compute mode to a value that includes the local environment (for example, All). If you do not, the local environment tree is not copied to the output message.

The information that you insert into DestinationData or Defaults depends on the characteristic of the corresponding node property:

- If a node property is represented by a check box (for example, New Message ID), set the Defaults or DestinationData element to Yes (equivalent to selecting the check box) or No (equivalent to clearing the check box).
- If a node property is represented by a drop-down list (for example, Transaction Mode), set the Defaults or DestinationData element to the appropriate character string (for example Automatic).
- If a node property is represented by a text entry field (for example, Queue Manager Name), set the Defaults or DestinationData element to the character string that you would enter in this field.

If necessary, configure the sending node to indicate where the destination information is. For example, for the output node MQOutput, set Destination Mode:

- If you set Destination Mode to Queue Name, the output message is sent to the queue identified in the output node properties Queue Name and Queue Manager Name. Destination is not referenced by the node.

- If you set Destination Mode to Destination List, the node extracts the destination information from the Destination subtree. If you use this value you can send a single message to multiple destinations, if you configure Destination and a single output node correctly. The node checks the node properties only if a value is not available in Destination (as described above).
- If you set Destination Mode to Reply To Queue, the message is sent to the reply-to queue identified in the MQMD in this message (field ReplyToQ). Destination is not referenced by the node.

To find more information about ESQL procedures that perform typical updates to the local environment, see "Populating Destination in the local environment tree" on page 355. Review the ESQL statements in these procedures to see how to modify the local environment. You can use these procedures unchanged, or modify them for your own requirements.

To find more information about how to extend the contents of this tree for your own purposes, see "Using scratchpad areas in the local environment."

For another example of how you can use the local environment to modify the behavior of a message flow, refer to the XML_PassengerQuery message flow in the following sample program:

- Airline Reservations

The Compute node in this message flow writes a list of destinations in the RouterList subtree of Destination that are used as labels by a later RouteToLabel node that propagates the message to the corresponding Label node. You can view samples information only when you use the information center that is integrated with the Message Broker Toolkit or the online information center.

Using scratchpad areas in the local environment:

The local environment tree includes a subtree called variables. This subtree is always created, but is never populated by the message flow. Use this area for your own purposes; for example, to pass information from one node to another. You can create other subtrees of the local environment tree.

The advantage of creating your own data in a scratchpad in the local environment is that this data can be propagated as part of the logical tree to subsequent nodes in the message flow. If you create a new output message in a Compute node, you can also include all or part of the local environment tree from the input message in the new output message.

To ensure that the information in the local environment is propagated further down the flow, the Compute mode property of the Compute node must be set to include the local environment as part of the output tree (for example, specify LocalEnvironment and Message). For further details about the Compute mode property, see "Setting the mode" on page 897.

However, any data updates or additions that you make in one node are not retained if the message moves backwards through the message flow (for example, if an exception is thrown). If you create your own data, and want that data to be preserved throughout the message flow, you must use the environment tree.

You can set values in the variables subtree in a Compute node and those values can be used later by another node (Compute, Database, or Filter) for some purpose that you determine when you configure the message flow.

The local environment is not in scope in a Compute node, therefore you must use `InputLocalEnvironment` and `OutputLocalEnvironment` instead. For example, you might use the scratchpad in the local environment to propagate the destination of an output message to subsequent nodes in a message flow. Your first Compute node determines that the output messages from this message flow must go to WebSphere MQ queues. Include the following ESQL to insert this information into the local environment by setting the value of *OutputLocation* in the OutputLocalEnvironment:

```
SET OutputLocalEnvironment.Variables.OutputLocation = 'MQ';
```

Your second Compute node can access this information from its input message. In the ESQL in this node, use the correlation name InputLocalEnvironment to identify the local environment tree in the input message that contains this data. The following ESQL sets *queueManagerName* and *queueName* based on the content of *OutputLocation* in the local environment, by using InputLocalEnvironment:

```
IF InputLocalEnvironment.Variables.OutputLocation = 'MQ' THEN
   SET OutputLocalEnvironment.Destination.MQ.DestinationData.queueManagerName = 'myQManagerName';
   SET OutputLocalEnvironment.Destination.MQ.DestinationData.queueName = 'myQueueName';
END IF;
```

In the example, *queueManagerName* and *queueName* are set for the Destination subtree in the output message. You must set the Compute mode of the second Compute node to include the local environment tree in the output message. Configure the MQOutput node to use the destination list that you have created in the local environment tree by setting the Destination Mode property to Destination List.

For information about the full list of elements in the DestinationData subtree, see "Data types for elements in the MQ DestinationData subtree" on page 1519.

Populating Destination in the local environment tree:

Use the Destination subtree to set up the target destinations that are used by output nodes, the HTTPRequest node, the SOAPRequest node, the SOAPAsyncRequest node, and the RouteToLabel node. The following examples show how you can create and use an ESQL procedure to perform the task of setting up values for each of these uses.

Copy and use these procedures as shown, or you can modify or extend them to perform similar tasks.

If you are creating this ESQL code for a Compute node, you must configure the node by setting the Compute Mode property so that it has access to the local environment tree in the output message. You must select one of the three values LocalEnvironment, LocalEnvironment And Message, or All.

Adding a queue name for the MQOutput node with the Destination Mode property set to Destination List

```
CREATE PROCEDURE addToMQDestinationList(IN LocalEnvironment REFERENCE, IN newQueue char) BEGIN
  /******************************************************************************
  * A procedure that adds a queue name to the MQ destination list in the local environment.
  * This list is used by an MQOutput node that has its mode set to Destination list.
  *
  * IN LocalEnvironment: the LocalEnvironment to be modified.
  * IN queue: the queue to be added to the list
  *
  ******************************************************************************/
DECLARE I INTEGER CARDINALITY(LocalEnvironment.Destination.MQ.DestinationData[]);
```

```
  IF I = 0 THEN
   SET OutputLocalEnvironment.Destination.MQ.DestinationData[1].queueName = newQueue;
  ELSE
   SET OutputLocalEnvironment.Destination.MQ.DestinationData[I+1].queueName = newQueue;
  END IF;
 END;
```

For full details of these elements, see "Data types for elements in the MQ DestinationData subtree" on page 1519.

Changing the default URL for a SOAPRequest node or a SOAPAsyncRequest node request

```
CREATE PROCEDURE overrideDefaultSOAPRequestURL(IN LocalEnvironment REFERENCE, IN newUrl char) BEGIN
  /*****************************************************************************
  * A procedure that changes the URL to which the SOAPRequest node or
  * SOAPAsyncRequest node sends the request.
  *
  * IN LocalEnvironment: the LocalEnvironment to be modified.
  * IN queue:  the URL to which to send the request.
  *
  *****************************************************************************/
SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.WebServiceURL = newUrl;
END;
```

Changing the default URL for an HTTPRequest node request

```
CREATE PROCEDURE overrideDefaultHTTPRequestURL(IN LocalEnvironment REFERENCE, IN newUrl char) BEGIN
  /*****************************************************************************
  * A procedure that changes the URL to which the HTTPRequest node sends the request.
  *
  * IN LocalEnvironment: the LocalEnvironment to be modified.
  * IN queue: the URL to which to send the request.
  *
  *****************************************************************************/
  SET OutputLocalEnvironment.Destination.HTTP.RequestURL = newUrl;
END;
```

Adding a label for the RouteToLabel node

```
CREATE PROCEDURE addToRouteToLabelList(IN LocalEnvironment REFERENCE, IN newLabel char) BEGIN
  /*****************************************************************************
  * A procedure that adds a label name to the RouteToLabel list in the local environment.
  * This list is used by a RoteToLabel node.
  *
  * IN LocalEnvironment: the LocalEnvironment to be modified.
  * IN label:  the label to be added to the list
  *
  *****************************************************************************/
 IF LocalEnvironment.Destination.RouterList.DestinationData is null THEN
     SET OutputLocalEnvironment.Destination.RouterList.DestinationData."label" = newLabel;
  ELSE
     CREATE LASTCHILD OF LocalEnvironment.Destination.RouterList.DestinationData
     NAME 'label' VALUE newLabel;
  END IF;
END;
```

Setting up JMS destination lists

You can configure a JMSOutput node to send to multiple JMS Queues, or to publish to multiple JMS Topics by using a destination list that is created in the local environment tree by a transformation node. The following example shows how to set up JMS destination lists in the local environment tree:

```
CREATE PROCEDURE CreateJMSDestinationList() BEGIN
  SET OutputLocalEnvironment.Destination.JMSDestinationList.DestinationData[1]  = 'jndi://TestDestQueue1';
  SET OutputLocalEnvironment.Destination.JMSDestinationList.DestinationData[2]  = 'jndi://TestDestQueue2';
  SET OutputLocalEnvironment.Destination.JMSDestinationList.DestinationData[3]  = 'jndi://TestDestQueue3';
END;
```

Accessing the environment tree:

The environment tree has its own correlation name, Environment, and you must use this name in all ESQL statements that refer to, or set, the content of this tree.

The environment tree is always created when the logical tree is created for an input message. However, the message flow neither populates it, nor uses its contents. You can use this tree for your own purposes, for example, to pass information from one node to another. You can use the whole tree as a scratchpad or working area.

The advantage of creating your own data in environment is that this data is propagated as part of the logical tree to subsequent nodes in the message flow. If you create a new output message in a Compute node, the environment tree is also copied from the input message to the new output message. (In contrast to the local environment tree, which is only included in the output message if you explicitly request that it is).

Only one environment tree is present for the duration of the message flow. Any data updates, or additions, that you make in one node are retained, and all of the nodes in the message flow have access to the latest copy of this tree. Even if the message flows back through the message flow (for example, if an exception is thrown, or if the message is processed through the second terminal of the FlowOrder node), the latest state is retained. (In contrast to the local environment tree, which reverts to its previous state if the message flows back through the message flow.)

You can use this tree for any purpose you choose. For example, you can use the following ESQL statements to create fields in the tree:

```
SET Environment.Variables =
      ROW('granary' AS bread, 'reisling' AS wine, 'stilton' AS cheese);
SET Environment.Variables.Colors[] =
      LIST{'yellow', 'green', 'blue', 'red', 'black'};
SET Environment.Variables.Country[] = LIST{ROW('UK' AS name, 'pound' AS currency),
     ROW('USA' AS name, 'dollar' AS currency)};
```

This information is now available to all nodes to which a message is propagated, regardless of their relative position in the message flow.

For another example of how you can use environment to store information used by other nodes in the message flow, look at the Reservation message flow in the following sample:

- Airline Reservations

The Compute node in this message flow writes information to the subtree Environment.Variables that it has extracted from a database according to the value of a field in the input message.

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

Accessing the ExceptionList tree using ESQL:

The ExceptionList tree has its own correlation name, ExceptionList, and you must use this in all ESQL statements that refer to or set the content of this tree.

This tree is created with the logical tree when an input message is parsed. It is initially empty, and is only populated if an exception occurs during message flow processing. It is possible that more than one exception can occur; if more than one exception occurs, the ExceptionList tree contains a subtree for each exception.

You can access the ExceptionList tree in Compute, Database, and Filter nodes, and you can update it in a Compute node. You must use the appropriate correlation name; ExceptionList for a Database or Filter node, and InputExceptionList for a Compute node.

You might want to access this tree in a node in an error handling procedure. For example, you might want to route the message to a different path based on the type of exception, for example one that you have explicitly generated using an ESQL THROW statement, or one that the broker has generated.

The following ESQL shows how you can access the ExceptionList and process each child that it contains:

```
-- Declare a reference for the ExceptionList
-- (in a Compute node use InputExceptionList)
DECLARE start REFERENCE TO ExceptionList.*[1];

-- Loop through the exception list children
WHILE start.Number IS NOT NULL DO
   -- more ESQL

   -- Move start to the last child of the field to which it currently points
   MOVE start LASTCHILD;
END WHILE;
```

The following example shows an extract of ESQL that has been coded for a Compute node to loop through the exception list to the last (nested) exception description and extract the error number. This error relates to the original cause of the problem and normally provides the most precise information. Subsequent action taken by the message flow can be decided by the error number retrieved in this way.

```
CREATE PROCEDURE getLastExceptionDetail(IN InputTree reference,OUT messageNumber integer,
OUT messageText char)
    /*****************************************************************************
  * A procedure that will get the details of the last exception from a message
  * IN InputTree:  The incoming exception list
  * IN messageNumber:  The last message numberr.
  * IN messageText: The last message text.
  *****************************************************************************/
   BEGIN
        -- Create a reference to the first child of the exception list
        declare ptrException reference to InputTree.*[1];
        -- keep looping while the moves to the child of exception list work
  WHILE lastmove(ptrException) DO
   -- store the current values for the error number and text
   IF ptrException.Number is not null THEN
          SET messageNumber = ptrException.Number;
          SET messageText = ptrException.Text;
     END IF;
     -- now move to the last child which should be the next exceptionlist
   move ptrException lastchild;
  END WHILE;
 END;
```

For more information about the use of ExceptionList, look at the subflow in the following sample which includes ESQL that interrogates the ExceptionList structure and takes specific action according to its content:

- Error Handler

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

For information on accessing the ExceptionList tree using Java, see "Accessing the ExceptionList tree using Java" on page 538

Transforming from one data type to another

Code ESQL functions and statements to transform messages and data types in many ways.

The following topics provide guidance:

- "Casting data from message fields"
- "Converting code page and message encoding" on page 360
- "Converting EBCDIC NL to ASCII CR LF" on page 362
- "Changing message format" on page 364

Casting data from message fields:

You can use the CAST function to transform the data type of one value to match the data type of the other. For example, you can use the CAST function when you process generic XML messages. All fields in an XML message have character values, so if you want to perform arithmetic calculations or datetime comparisons, for example, you must convert the string value of the field into a value of the appropriate type using CAST.

When you compare an element with another element, variable or constant, ensure that the value with which you are comparing the element is consistent (for example, character with character). If the values are not consistent, the broker generates a runtime error if it cannot provide an implicit casting to resolve the inconsistency. For details of what implicit casts are supported, see Implicit casts.

In the Invoice message, the field InvoiceDate contains the date of the invoice. If you want to refer to or manipulate this field, you must CAST it to the correct format first. For example, to refer to this field in a test:

```
IF CAST(Body.Invoice.InvoiceDate AS DATE) = CURRENT_DATE THEN
```

This converts the string value of the InvoiceDate field into a date value, and compares it to the current date.

Another example is casting from integer to character:

```
DECLARE I INTEGER 1;
DECLARE C CHARACTER;

-- The following statement generates an error
SET C = I;

-- The following statement is valid
SET C = CAST(I AS CHARACTER);
```

Converting code page and message encoding:

You can use ESQL within a Compute node to convert data for code page and message encoding.

If your message flow is processing WebSphere MQ messages, you can use WebSphere MQ facilities (including get and put options and WebSphere MQ data conversion exits) to provide these conversions. If you are not processing WebSphere MQ messages, or you choose not to use WebSphere MQ facilities, you can use WebSphere Message Broker facilities by coding the appropriate ESQL in a Compute node in your message flow.

The contents of the MQMD, the MQRFH2, and the message body of a message in the MRM domain that has been modeled with a CWF physical format can be subject to code page and encoding conversion. The contents of a message body of a message in the XML, XMLNS, and JMS domains, and those messages in the MRM domain that have been modeled with an XML or TDS physical format, are treated as strings. Only code page conversion applies; no encoding conversion is required.

For messages in the MRM domain modeled with a CWF physical format, you can set the MQMD CCSID and Encoding fields of the output message, plus the CCSID and Encoding of any additional headers, to the required target value.

For messages in the MRM domain modeled with an XML or TDS physical format, you can set the MQMD CCSID field of the output message, plus the CCSID of any additional headers. XML and TDS data is handled as strings and is therefore subject to CCSID conversion only.

An example WebSphere MQ message has an MQMD header, an MQRFH2 header, and a message body. To convert this message to a mainframe CodedCharSetId and Encoding, code the following ESQL in the Compute node:

```
SET OutputRoot.MQMD.CodedCharSetId = 500;
SET OutputRoot.MQMD.Encoding = 785;
SET OutputRoot.MQRFH2.CodedCharSetId = 500;
SET OutputRoot.MQRFH2.Encoding = 785;
```

The following example illustrates what you must do to modify a CWF message so that it can be passed from WebSphere Message Broker to IMS on z/OS.

1. You have defined the input message in XML and are using an MQRFH2 header. Remove the header before passing the message to IMS.
2. The message passed to IMS must have MQIIH header, and must be in the z/OS code page. This message is modeled in the MRM and has the name IMS1. Define the PIC X fields in this message as logical type string for conversions between EBCDIC and ASCII to take place. If the fields are binary logical type, no data conversion occurs; binary data is ignored when a CWF message is parsed by the MRM parser.
3. The message received from IMS is also defined in the MRM and has the name IMS2. Define the PIC X fields in this message as logical type string for conversions between EBCDIC and ASCII to take place. If the fields are binary logical type, no data conversion occurs; binary data is ignored when a CWF message is parsed by the MRM parser.
4. Convert the reply message to the Windows code page. The MQIIH header is retained on this message.

5. You have created a message flow that contains the following nodes: :
   a. The outbound flow, **MQInput1 --> Compute1 --> MQOutput1**.
   b. The inbound flow, **MQInput2 --> Compute2 --> MQOutput2**.
6. Code ESQL in Compute1 (outbound) node as follows, specifying the relevant MessageSet ID. This code shows the use of the default CWF physical layer name. You must use the name that matches your model definitions. If you specify an incorrect value, the broker fails with message BIP5431.

```
-- Loop to copy message headers
DECLARE I INTEGER 1;
DECLARE J INTEGER CARDINALITY(InputRoot.*[]);

WHILE I < J - 1 DO
  SET OutputRoot.*[I] = InputRoot.*[I];
  SET I=I+1;
END WHILE;

SET OutputRoot.MQMD.CodedCharSetId = 500;
SET OutputRoot.MQMD.Encoding = 785;
SET OutputRoot.MQMD.Format = 'MQIMS   ';
SET OutputRoot.MQIIH.Version = 1;
SET OutputRoot.MQIIH.StrucLength = 84;
SET OutputRoot.MQIIH.Encoding = 785;
SET OutputRoot.MQIIH.CodedCharSetId = 500;
SET OutputRoot.MQIIH.Format = 'MQIMSVS ';
SET OutputRoot.MQIIH.Flags = 0;
SET OutputRoot.MQIIH.LTermOverride = '        ';
SET OutputRoot.MQIIH.MFSMapName = '        ';
SET OutputRoot.MQIIH.ReplyToFormat = 'MQIMSVS ';
SET OutputRoot.MQIIH.Authenticator = '        ';
SET OutputRoot.MQIIH.TranInstanceId = X'00000000000000000000000000000000';
SET OutputRoot.MQIIH.TranState = ' ';
SET OutputRoot.MQIIH.CommitMode = '0';
SET OutputRoot.MQIIH.SecurityScope = 'C';
SET OutputRoot.MQIIH.Reserved = ' ';
SET OutputRoot.MRM.e_elen08 = 30;
SET OutputRoot.MRM.e_elen09 = 0;
SET OutputRoot.MRM.e_string08 = InputBody.e_string01;
SET OutputRoot.MRM.e_binary02 = X'31323334353637383940';
SET OutputRoot.Properties.MessageSet = 'DHCJOEG072001';
SET OutputRoot.Properties.MessageType = 'IMS1';
SET OutputRoot.Properties.MessageFormat = 'Binary1';
```

   The use of a variable, J, that is initialized to the value of the cardinality of the existing headers in the message, is more efficient than calculating the cardinality on each iteration of the loop, which happens if you code the following WHILE statement:

```
WHILE I < CARDINALITY(InputRoot.*[]) DO
```

7. Create ESQL in Compute2 (inbound) node as follows, specifying the relevant MessageSet ID. This code shows the use of the default CWF physical layer name. You must use the name that matches your model definition. If you specify an incorrect value, the broker fails with message BIP5431.

```
-- Loop to copy message headers
DECLARE I INTEGER 1;
DECLARE J INTEGER CARDINALITY(InputRoot.*[]);

WHILE I < J DO
  SET OutputRoot.*[I] = InputRoot.*[I];
  SET I=I+1;
END WHILE;

SET OutputRoot.MQMD.CodedCharSetId = 437;
SET OutputRoot.MQMD.Encoding = 546;
SET OutputRoot.MQMD.Format = 'MQIMS   ';
SET OutputRoot.MQIIH.CodedCharSetId = 437;
SET OutputRoot.MQIIH.Encoding = 546;
SET OutputRoot.MQIIH.Format = '        ';
SET OutputRoot.MRM = InputBody;
SET OutputRoot.Properties.MessageSet = 'DHCJOEG072001';
SET OutputRoot.Properties.MessageType = 'IMS2';
SET OutputRoot.Properties.MessageFormat = 'Binary1';
```

You do not have to set any specific values for the MQInput1 node properties, because the message and message set are identified in the MQRFH2 header, and no conversion is required.

You must set values for message domain, set, type, and format in the MQInput node for the inbound message flow (MQInput2). You do not need to set conversion parameters.

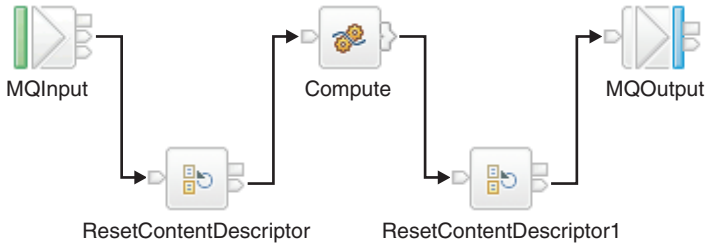
One specific situation in which you might need to convert data in one code page to another is when messages contain newline characters and are passed between EBCDIC and ASCII systems. The required conversion for this situation is described in “Converting EBCDIC NL to ASCII CR LF.”

Converting EBCDIC NL to ASCII CR LF:

You might want to change newline (NL) characters in a text message to carriage return (CR) and line feed (LF) character pairs. This example shows one way in which you can convert these characters.

This conversion might be useful if messages from an EBCDIC platform (for example, using CCSID 1047) are sent to an ASCII platform (for example, using CCSID 437). Problems can arise because the EBCDIC NL character hex '15' is converted to the undefined ASCII character hex '7F'. The ASCII code page has no corresponding code point for the NL character.

In this example, a message flow is created that interprets the input message as a message in the BLOB domain. This message is passed into a ResetContentDescriptor node to reset the data to a message in the MRM domain. The message is called msg_nl (a set of repeating string elements delimited by EBCDIC NL characters). A Compute node is then used to create an output based on another message in the MRM domain called msg_crlf (a set of repeating string elements delimited by CR LF pairs). The message domain is then changed back to BLOB in another ResetContentDescriptor node. This message flow is shown in the following diagram.



The following instructions show how to create the messages and configure the message flow.

1. Create the message models for the messages in the MRM domain:
   a. Create a message set project called myProj.
   b. Create a message set called myMessageSet with a TDS physical format (the default name is Text1).
   c. Create an element string1 of type xsd:string.
   d. Create a complex type called t_msg_nl and specify the following complex type properties:
      - Composition = Ordered Set
      - Content Validation = Closed
      - Data Element Separation = All Elements Delimited
      - Delimiter = <U+0085> (hex '0085' is the UTF-16 representation of an NL character)
      - Repeat = Yes
      - Min Occurs = 1
      - Max Occurs = 50 (the text of the message is assumed to consist of no more than 50 lines)
   e. Add Element string1, and set the following property:
      - Repeating Element Delimiter = <U+0085>
   f. Create a Message msg_nl, and set its associated complex type to t_msg_nl
   g. Create a complex type called t_msg_crlf, and specify the following complex type properties:
      - Composition = Ordered Set
      - Content Validation = Closed
      - Data Element Separation = All Elements Delimited
      - Delimiter = <CR><LF> (<CR> and <LF> are the mnemonics for the CR and LF characters)
      - Repeat = Yes
      - Min Occurs = 1
      - Max Occurs = 50
   h. Add Element string1, and set the following property:
      - Repeating Element Delimiter = <CR><LF>
   i. Create a Message msg_crlf, and set complex type to t_msg_crlf.
2. Configure the message flow shown in the previous figure:
   a. Start with the MQInput node:
      - Set Message Domain = BLOB
      - Set Queue Name = <Your input message queue name>
   b. Add the ResetContentDescriptor node, connected to the Out terminal of the MQInput node:
      - Set Message Domain = MRM

- Select Reset Message Domain
- Set Message Set = <Your Message Set ID> (this field has a maximum of 13 characters)
- Select Reset Message Set
- Set Message Type = msg_nl
- Select Reset Message Type
- Set Message Format = Text1
- Select Reset Message Format

c. Add the Compute node, connected to the Out terminal of the ResetContentDescriptor node:
   - Enter a name for the ESQL Module for this node, or accept the default (<message flow name>_Compute).
   - Right-click the Compute node, and select **Open ESQL**. Add the following ESQL code in the module:

```
-- Declare local working variables
DECLARE I INTEGER 1;
DECLARE J INTEGER CARDINALITY(InputRoot.*[]);

-- Loop to copy all message headers from input to output message
WHILE I < J DO
  SET OutputRoot.*[I] = InputRoot.*[I];
  SET I=I+1;
END WHILE;

-- Set new output message type which uses CRLF delimiter
SET OutputRoot.Properties.MessageType = 't_msg_crlf';

-- Loop to copy each instance of string1 child within message body
SET I = 1;
SET J = CARDINALITY("InputBody"."string1"[]);
WHILE I <= J DO
  SET "OutputRoot"."MRM"."string1"[I] = "InputBody"."string1"[I];
  SET I=I+1;
END WHILE;
```

The use of a variable, J, initialized to the value of the cardinality of the existing headers in the message, is more efficient than calculating the cardinality on each iteration of the loop, which happens if you code the following WHILE statement:

```
WHILE I < CARDINALITY(InputRoot.*[]) DO
```

d. Add the ResetContentDescriptor1 node, connected to the Out terminal of the Compute node:
   - Set Message Domain = BLOB
   - Select Reset Message Domain.

e. Finally, add the MQOutput node, connected to the Out terminal of the ResetContentDescriptor1 node. Configure its properties to direct the output message to the required queue or queues.

Changing message format:

Use the Compute node to copy part of an input message to an output message. The results of such a copy depend on the type of input and output parsers involved.

Like parsers:

Where both the source and target messages have the same folder structure at root level, a *like-parser-copy* is performed. For example:

```
SET OutputRoot.MQMD = InputRoot.MQMD;
```

This statement copies all the children in the MQMD folder of the input message to the MQMD folder of the output message.

Another example of a tree structure that supports a like-parser-copy is:

```
SET OutputRoot.XMLNS.Data.Account = InputRoot.XMLNS.Customer.Bank.Data;
```

To transform an input message in the MRM domain to an output message also in the MRM domain, you can use either the Compute or the Mapping node. The Mapping node can interpret the action that is required because it knows the format of both messages. Content Assist in the ESQL module for the Compute node can also use the message definitions for those messages. If the messages are not in the same namespace, you must use the Compute node.

To use Content Assist with message references, you must set up a project reference from the project containing the ESQL to the project containing the message set. For information about setting up a project reference, see Project references.

If both input and output messages are not in the MRM domain, you must use the Compute node and specify the structure of the messages yourself.

Unlike parsers:

Where the source and target messages have different folder structures at root level, you cannot make an exact copy of the message source. Instead, the *unlike-parser-copy* views the source message as a set of nested folders terminated by a leaf name-value pair. For example, copying the following message from XML to MRM:

```
<Name3><Name31>Value31</Name31>Value32</Name3>
```

produces a name element Name3, and a name-value element called Name31 with the value Value31. The second XML pcdata (Value32) cannot be represented and is discarded.

The unlike-parser-copy scans the source tree, and copies folders, also known as name elements, and leaf name-value pairs. Everything else, including elements flagged as *special* by the source parser, is not copied.

An example of a tree structure that results in an unlike-parser-copy is:

```
SET OutputRoot.MRM.Data.Account = InputRoot.XMLNS.Data.Account;
```

If the algorithm used to make an unlike-parser-copy does not suit your tree structure, you should further qualify the source field to restrict the amount of the tree that is copied.

Be careful when you copy information from input messages to output messages in different domains. You could code ESQL that creates a message structure or content that is not completely consistent with the rules of the parser that processes the output message. This action can result in an output message not being created, or being created with unexpected content. If you believe that the output message

generated by a particular message flow does not contain the correct content, or have the expected form, check the ESQL that creates the output message, and look for potential mismatches of structure, field types, field names, and field values.

When copying trees between unlike parsers, you should set the message format of the target parser. For example, if a message set has been defined with XMLNS and CWF formats, the following commands are required to copy an input XMLNS stream to the MRM parser and set the latter to be generated in CWF format:

```
-- Copy message to the output, moving from XMLNS to MRM domains
SET OutputRoot.MRM = InputRoot.XMLNS.rootElement;

-- Set the CWF format for output by the MRM domain
SET OutputRoot.Properties.MessageType = '<MessageTypeName>';
SET OutputRoot.Properties.MessageSet = '<MessageSetName>';
SET OutputRoot.Properties.MessageFormat = 'CWF';
```

Adding keywords to ESQL files

You can add keywords to ESQL files to contain information that you want to associate with a message flow.

Use one or more of the following methods:

Comment fields

Add the keyword as a comment in the ESQL file:

```
-- $MQSI compiled by = John MQSI$
```

Static strings

Include the keyword as part of a static string in the ESQL file:

```
SET target = '$MQSI_target = production only MQSI$'
```

Variable string

Include the keyword value as a variable string in the ESQL file:

```
$MQSI_VERSION=$id$MQSI$
```

In this example, when the message flow source is extracted from the file repository, the repository's plug-in has been configured to substitute the identifier *id* with the actual version number. The identifier value that is required depends on the capability and configuration of the repository, and is not part of WebSphere Message Broker.

Restrictions within keywords

Do not use the following characters within keywords, because they cause unpredictable behavior:

```
^ $ . | \ < > ? + * = & [ ] ( )
```

You can use these characters in the values that are associated with keywords; for example:

- `$MQSI RCSVER=$id$ MQSI$` is acceptable
- `$MQSI $name=Fred MQSI$` is not acceptable

Interaction with databases using ESQL

Use ESQL statements and functions to read from, write to, and modify databases from your message flows.

ESQL has a number of statements and functions for accessing databases:

- The CALL statement invokes a stored procedure.
- The DELETE FROM statement removes rows from a database table.
- The INSERT statement adds a row to a database table.
- The PASSTHRU function can be used to make complex selections.
- The PASSTHRU statement can be used to invoke administrative operations (for example, creating a table).
- The SELECT function retrieves data from a table.
- The UPDATE statement changes one or more values stored in zero or more rows.

You can access user databases from Compute, Database, and Filter nodes. The database access capabilities of all these nodes is identical.

You can use the data in the databases to update or create messages, or use the data in the message to update or create data in the databases.

Select **Throw exception on database error** and **Treat warnings as errors**, and set **Transaction** to **Automatic** on each node that access a database, to provide maximum flexibility.

For information about configuring the broker and the database to support access from message flows, see "Accessing databases from ESQL" on page 212.

Referencing columns in a database:

While the standard SQL SELECT syntax is supported for queries to an external database, there are a number of points to be borne in mind. You must prefix the name of the table with the keyword Database to indicate that the SELECT is to be targeted at the external database, rather than at a repeating structure in the message.

The basic form of database SELECT is:

```
SELECT ...
  FROM Database.TABLE1
  WHERE ...
```

If necessary, you can specify a schema name:

```
SELECT ...
  FROM Database.SCHEMA.TABLE1
  WHERE ...
```

where SCHEMA is the name of the schema in which the table TABLE1 is defined. Include the schema if the user ID under which you are running does not match the schema. For example, if your userID is USER1, the expression Database.TABLE1 is equivalent to Database.USER1.TABLE1. However, if the schema associated with the table in the database is db2admin, you must specify Database.db2admin.TABLE1.

If you do not include the schema, and this does not match your current user ID, the broker generates a runtime error when a message is processed by the message flow.

If, as in the two previous examples, a data source is not specified, TABLE1 must be a table in the default database specified by the node's `data source` property. To access data in a database other than the default specified on the node's `data source` property, you must specify the data source explicitly. For example:

```
SELECT ...
  FROM Database.DataSource.SCHEMA.TABLE1
  WHERE ...
```

Qualify references to column names with either the table name or the correlation name defined for the table by the FROM clause. So, where you could normally execute a query such as:

```
SELECT column1, column2 FROM table1
```

you must write one of the following two forms:

```
SELECT T.column1, T.column2 FROM Database.table1 AS T

SELECT table1.column1, table1.column2 FROM Database.table1
```

This is necessary in order to distinguish references to database columns from any references to fields in a message that might also appear in the SELECT:

```
SELECT T.column1, T.column2 FROM Database.table1
  AS T WHERE T.column3 = Body.Field2
```

You can use the AS clause to rename the columns returned. For example:

```
SELECT T.column1 AS price, T.column2 AS item
  FROM Database.table1 AS T WHERE...
```

The standard select all SQL option is supported in the SELECT clause. If you use this option, you must qualify the column names with either the table name or the correlation name defined for the table. For example:

```
SELECT T.* FROM Database.Table1 AS T
```

When you use ESQL procedure and function names within a database query, the positioning of these within the call affects how these names are processed. If it is determined that the procedure or function affects the results returned by the query, it is not processed as ESQL and is passed as part of the database call.

This applies when attempting to use a function or procedure name with the column identifiers within the SELECT statement.

For example, if you use a CAST statement on a column identifier specified in the Select clause, this is used during the database query to determine the data type of the data being returned for that column. An ESQL CAST is not performed to that ESQL data type, and the data returned is affected by the database interaction's interpretation of that data type.

If you use a function or procedure on a column identifier specified in the WHERE clause, this is passed directly to the database manager for processing.

The examples in the subsequent topics illustrate how the results sets of external database queries are represented in WebSphere Message Broker. The results of database queries are assigned to fields in a message using a Compute node.

A column function is a function that takes the values of a single column in all the selected rows of a table or message and returns a single scalar result.

Selecting data from database columns:

You can configure a Compute, Filter, or Database node to select data from database columns and include it in an output message.

The following example assumes that you have a database table called USERTABLE with two char(6) data type columns (or equivalent), called Column1 and Column2. The table contains two rows:

	Column1	**Column2**
Row 1	value1	value2
Row 2	value3	value4

Configure the Compute, Filter, or Database node to identify the database in which you have defined the table. For example, if you are using the default database (specified on the `data source` property of the node), right-click the node, select **Open ESQL**, and code the following ESQL statements in the module for this node:

```
SET OutputRoot = InputRoot;
DELETE FIELD OutputRoot.*[<];
SET OutputRoot.XML.Test.Result[] =
  (SELECT T.Column1, T.Column2 FROM Database.USERTABLE AS T);
```

This ESQL produces the following output message:

```
<Test>
   <Result>
      <Column1>value1</Column1>
      <Column2>value2</Column2>
   </Result>
   <Result>
      <Column1>value3</Column1>
      <Column2>value4</Column2>
   </Result>
</Test>
```

Figure 3. Output message

To trigger the SELECT, send a trigger message with an XML body that is of the following form:

```
<Test>
   <Result>
      <Column1></Column1>
      <Column2></Column2>
   </Result>
   <Result>
      <Column1></Column1>
      <Column2></Column2>
   </Result>
</Test>
```

The exact structure of the XML is not important, but the enclosing tag must be <Test> to match the reference in the ESQL. If the enclosing tag is not <Test>, the ESQL statements result in top-level enclosing tags being formed, which is not valid XML.

If you want to create an output message that includes all the columns of all the rows that meet a particular condition, use the SELECT statement with a WHERE clause:

```
-- Declare and initialize a variable to hold the
--      test vaue (in this case the surname Smith)
DECLARE CurrentCustomer STRING 'Smith';

-- Loop through table records to extract matching information
SET OutputRoot.XML.Invoice[] =
    (SELECT R FROM Database.USERTABLE AS R
              WHERE R.Customer.LastName = CurrentCustomer
    );
```

The message fields are created in the same order as the columns occur in the table.

If you are familiar with SQL in a database environment, you might expect to code SELECT *. This syntax is not accepted by the broker, because you must start all references to columns with a correlation name to avoid ambiguities with declared variables. Also, if you code SELECT I.*, this syntax is accepted by the broker, but the * is interpreted as the first child element, not all elements, as you might expect from other database SQL.

The assignment of the result set of a database into a parser-owned message tree requires the result set to exactly match the message definition. Because the generic XML parser is self-defining, the example creates a new subtree off the Invoice folder, and the parser can parse the new elements in the subtree. If the structure of the result set exactly matches the message definition, the result set can be assigned directly into the OutputRoot message body tree.

If the structure of the result set does not exactly match the MRM message definition, you must first assign the result set into a ROW data type, or an Environment tree that does not have a parser associated with it.

The required data can then be assigned to OutputRoot to build a message tree that conforms to the message definition.

Selecting data from a table in a case-sensitive database system:

If the database system is case sensitive, you must use an alternative approach. This approach is also necessary if you want to change the name of the generated field to something different:

```
SET OutputRoot = InputRoot;
SET OutputRoot.XML.Test.Result[] =
  (SELECT T.Column1 AS Column1, T.Column2 AS Column2
  FROM Database.USERTABLE AS T);
```

This example produces the output message as shown in Figure 3 on page 369. Ensure that references to the database columns (in this example, T.Column1 and T.Column2) are specified in the correct case to match the database definitions exactly. If you do not match the database definitions exactly (for example if you specify T.COLUMN1), the broker generates a runtime error. Column1 and Column2

are used in the SELECT statement to match the columns that you have defined in the database, although you can use any values here; the values do not have to match.

Selecting bitstream data from a database:

These samples show how to retrieve XML bitstream data from a database, and include it in an output message. See INSERT statement for examples that show how you can insert bitstream data into a database.

In the following example, bitstream data is held in a database column with a BLOB data type. The database table used in the example (TABLE1) is the one created in the INSERT statement examples, and the table contains the following columns:

- MSGDATA
- MSGCCSID
- MSGENCODING

If the bit stream from the database does not need to be interrogated or manipulated by the message flow, the output message can be constructed in the BLOB domain without any alteration.

In the following example, the message data, along with the MQMD header, is held in a database column with a BLOB data type. To re-create the message tree, including the MQMD header, from the bit stream, you can use a CREATE statement with a PARSE clause and DOMAIN('MQMD'). The output message can then be modified by the message flow:

```
SET Environment.Variables.DBResult = THE( SELECT T.* FROM Database.TABLE1 AS T);
DECLARE resultRef REFERENCE TO Environment.Variables.DBResult;

IF LASTMOVE(resultRef) THEN

  DECLARE outMsg BLOB resultRef.MSGDATA ;
  DECLARE outCCSID INT resultRef.MSGCCSID;
  DECLARE outEncoding INT resultRef.MSGENCODING;
  DECLARE outMsgPriority INT resultRef.MSGPRIORITY;
  DECLARE outMsgSeqNum INT resultRef.MSGSEQNUMBER;

  SET OutputRoot.Properties.CodedCharSetId = outCCSID;
  SET OutputRoot.Properties.Encoding = outEncoding ;

  CREATE LASTCHILD OF OutputRoot DOMAIN('MQMD') PARSE(outMsg, outEncoding, outCCSID);

    SET OutputRoot.MQMD.Version = MQMD_VERSION_2;


  SET OutputRoot.MQMD.Priority = outMsgPriority;
  SET OutputRoot.MQMD.MsgSeqNumber = outMsgSeqNum;

  DECLARE HDRL INT ;
  SET HDRL = LENGTH(BITSTREAM(OutputRoot.MQMD));
  CREATE FIELD OutputRoot."BLOB"."BLOB";
  DECLARE MSGB BLOB;
  SET MSGB = SUBSTRING(outMsg FROM HDRL +1);
  SET OutputRoot."BLOB"."BLOB" = MSGB;

END IF;
```

If you want to interrogate or manipulate a bit stream extracted from a database, you must re-create the original message tree. To re-create the XML message tree

from the bit stream, you can use a CREATE statement with a PARSE clause. The output message can then be modified by the message flow.

For example, you might create a database table by using the following statement:

```
INSERT INTO Database.TABLE1(MSGDATA, MSGENCODING, MSGCCSID)
   VALUES (msgBitStream, inEncoding, inCCSID);
```

The following code snippet shows how to re-create the message tree in the XMLNS domain by using the data read from the table:

```
CALL CopyMessageHeaders();
  SET Environment.Variables.DBResult = THE( SELECT T.* FROM Database.TABLE1 AS T);
  DECLARE resultRef REFERENCE TO Environment.Variables.DBResult;
  IF LASTMOVE(resultRef) THEN
    DECLARE outCCSID INT resultRef.MSGCCSID;
    DECLARE outEncoding INT resultRef.MSGENCODING;
    DECLARE outMsg BLOB resultRef.MSGDATA;
    SET OutputRoot.Properties.CodedCharSetId = outCCSID;
    SET OutputRoot.Properties.Encoding = outEncoding;
    CREATE LASTCHILD OF OutputRoot DOMAIN('XMLNS') PARSE(outMsg, outEncoding, outCCSID);
    -- Now modify the message tree fields
    SET OutputRoot.XMLNS.A.B = 4;
    SET OutputRoot.XMLNS.A.E = 5;
  END IF;
```

In the following example, the data is held in a database column with a character data type, such as CHAR or VARCHAR. A cast is used to convert the data extracted from the database into BLOB format. If the bitstream data from the database does not need to be interrogated or manipulated by the message flow, the output message can be constructed in the BLOB domain, without any alteration.

```
CALL CopyMessageHeaders();
  SET Environment.Variables.DBResult = THE( SELECT T.* FROM Database.TABLE1 AS T);
  DECLARE resultRef REFERENCE TO Environment.Variables.DBResult;
  IF LASTMOVE(resultRef) THEN
    DECLARE outCCSID INT resultRef.MSGCCSID;
    DECLARE outMsg BLOB CAST(resultRef.MSGDATA AS BLOB CCSID outCCSID);
    SET OutputRoot.Properties.CodedCharSetId = outCCSID;
    SET OutputRoot.Properties.Encoding = resultRef.MSGENCODING;
    SET OutputRoot.BLOB.BLOB = outMsg;
  END IF;
```

In the following example, the data is held in a database column with a character data type, such as CHAR or VARCHAR. A cast is used to convert the data extracted from the database into BLOB format. To manipulate or interrogate this data within the message flow, you must re-create the original message tree. In this example, a CREATE statement with a PARSE clause is used to re-create the XML message tree in the XMLNS domain.

```
CALL CopyMessageHeaders();
  SET Environment.Variables.DBResult = THE( SELECT T.* FROM Database.TABLE1 AS T);
  DECLARE resultRef REFERENCE TO Environment.Variables.DBResult;
  IF LASTMOVE(resultRef) THEN
    DECLARE outCCSID INT resultRef.MSGCCSID;
    DECLARE outEncoding INT resultRef.MSGENCODING;
    DECLARE outMsg BLOB CAST(resultRef.MSGDATA AS BLOB CCSID outCCSID);
    SET OutputRoot.Properties.CodedCharSetId = outCCSID;
    SET OutputRoot.Properties.Encoding = outEncoding;
    CREATE LASTCHILD OF OutputRoot DOMAIN('XMLNS') PARSE(outMsg, outEncoding, outCCSID);
    -- Now modify the message tree fields
    SET OutputRoot.XMLNS.A.B = 4;
    SET OutputRoot.XMLNS.A.E = 5;
  END IF;
```

Accessing multiple database tables:

You can refer to multiple tables that you have created in the same database. Use the FROM clause on the SELECT statement to join the data from the two tables.

The following example assumes that you have two database tables called USERTABLE1 and USERTABLE2. Both tables have two char(6) data type columns (or equivalent).

USERTABLE1 contains two rows:

	Column1	**Column2**
Row 1	value1	value2
Row 2	value3	value4

USERTABLE2 contains two rows:

	Column3	**Column4**
Row 1	value5	value6
Row 2	value7	value8

All tables referenced by a single SELECT function must be in the same database. The database can be either the default (specified on the Data Source property of the node) or another database (specified on the FROM clause of the SELECT function).

Configure the Compute, Database, or Filter node that you are using to identify the database in which you have defined the tables. For example, if you are using the default database, right-click the node, select **Open ESQL**, and code the following ESQL statements in the module for this node:

```
SET OutputRoot.XML.Test.Result[] =
         (SELECT A.Column1 AS FirstColumn,
                 A.Column2 AS SecondColumn,
                 B.Column3 AS ThirdColumn,
                 B.Column4 AS FourthColumn
            FROM Database.USERTABLE1 AS A,
                 Database.USERTABLE2 AS B
           WHERE A.Column1 = 'value1' AND
                 B.Column4 = 'value8'
         );
```

This code results in the following output message content:

```
<Test>
  <Result>
    <FirstColumn>value1</FirstColumn>
    <SecondColumn>value2</SecondColumn>
    <ThirdColumn>value7</ThirdColumn>
    <FourthColumn>value8</FourthColumn>
  </Result>
</Test>
```

This example shows how to access data from two database tables. You can code more complex FROM clauses to access multiple database tables (although all the tables must be in the same database). You can also refer to one or more message

trees, and can use SELECT to join tables with tables, messages with messages, or tables with messages. “Joining data from messages and database tables” on page 395 provides an example of how to merge message data with data in a database table.

If you specify an ESQL function or procedure on the column identifier in the WHERE clause, it is processed as part of the database query and not as ESQL.

Consider the following example:

```
SET OutputRoot.XML.Test.Result =
   THE(SELECT ITEM T.Column1 FROM Database.USERTABLE1 AS T
   WHERE UPPER(T.Column2) = 'VALUE2');
```

This code attempts to return the rows where the value of `Column2` converted to uppercase is `VALUE2`. However, only the database manager can determine the value of `T.Column2` for any given row, therefore it cannot be processed by ESQL before the database query is issued, because the WHERE clause determines the rows that are returned to the message flow.

Therefore, the UPPER is passed to the database manager to be included as part of its processing. However, if the database manager cannot process the token within the SELECT statement, an error is returned.

Changing database content:

You can use Compute, Database, and Filter nodes to change the contents of a database by updating, inserting, or deleting data.

The following ESQL code includes statements that show all three operations. This code is appropriate for a Database and Filter node; if you create this code for a Compute node, use the correlation name InputRoot in place of Root.

```
IF Root.XMLNS.TestCase.Action = 'INSERT' THEN
   INSERT INTO Database.STOCK (STOCK_ID, STOCK_DESC, STOCK_QTY_HELD,
   BROKER_BUY_PRICE, BROKER_SELL_PRICE, STOCK_HIGH_PRICE, STOCK_HIGH_DATE,
   STOCK_HIGH_TIME) VALUES
   (CAST(Root.XMLNS.TestCase.stock_id AS INTEGER),
    Root.XMLNS.TestCase.stock_desc,
    CAST(Root.XMLNS.TestCase.stock_qty_held AS DECIMAL),
    CAST(Root.XMLNS.TestCase.broker_buy_price AS DECIMAL),
    CAST(Root.XMLNS.TestCase.broker_sell_price AS DECIMAL),
    Root.XMLNS.TestCase.stock_high_price,
    CURRENT_DATE,
    CURRENT_TIME);

ELSEIF Root.XMLNS.TestCase.Action = 'DELETE' THEN

        DELETE FROM Database.STOCK WHERE STOCK.STOCK_ID =
                    CAST(Root.XMLNS.TestCase.stock_id AS INTEGER);

   ELSEIF Root.XMLNS.TestCase.Action = 'UPDATE' THEN

          UPDATE Database.STOCK as A SET STOCK_DESC = Root.XMLNS.TestCase.stock_desc
                 WHERE  A.STOCK_ID = CAST(Root.XMLNS.TestCase.stock_id AS INTEGER);
END IF;
```

Checking returns to SELECT:

If a SELECT function returns no data, or no further data, this result is handled as a normal situation and no error code is set in SQLCODE, regardless of the setting of the Throw Exception On Database Error and Treat Warnings As Errors properties on the current node.

To recognize that a SELECT function has returned no data, include ESQL that checks what has been returned. You can use various methods:

1. EXISTS

   This ESQL returns a Boolean value that indicates if a SELECT function returned one or more values (TRUE), or none (FALSE).

   ```
   IF EXISTS(SELECT T.MYCOL FROM Database.MYTABLE) THEN
   ...
   ```

2. CARDINALITY

   If you expect an array in response to a SELECT, you can use CARDINALITY to calculate how many entries have been received.

   ```
   SET OutputRoot.XMLNS.Testcase.Results[] = (
       SELECT T.MYCOL FROM Database.MYTABLE)
   ......
   IF CARDINALITY (OutputRoot.XMLNS.Testcase.Results[])> 0 THEN
   ........
   ```

3. IS NULL

   If you have used either THE or ITEM keywords in your SELECT function, a scalar value is returned. If no rows have been returned, the value set is NULL. However, it is possible that the value NULL is contained within the column, and you might want to distinguish between these two cases.

   Distinguish between cases by including COALESCE in the SELECT function, for example:

   ```
   SET OutputRoot.XMLNS.Testcase.Results VALUE = THE (
      SELECT ITEM COALESCE(T.MYCOL, 'WAS NULL')
      FROM Database.MYTABLE);
   ```

   If this example returns the character string WAS NULL, it indicates that the column contained NULL, and not that no rows were returned.

In previous releases, an SQLCODE of 100 was set in most cases if no data, or no further data, was returned. An exception was raised by the broker if you chose to handle database errors in the message flow.

Committing database updates:

When you create a message flow that interacts with databases, you can choose whether the updates that you make are committed when the current node has completed processing, or when the current invocation of the message flow has terminated.

For each node, select the appropriate option for the Transaction property to specify when its database updates are to be committed:

- Choose Automatic (the default) if you want updates made in this node to be committed or rolled back as part of the whole message flow. The actions that you define in the ESQL module are performed on the message and it continues through the message flow. If the message flow completes successfully, the updates are committed. If the message flow fails, the message and the database updates are rolled back.
- Choose Commit if you want to commit the action of the node on the database, irrespective of the success or failure of the message flow as a whole. The

database update is committed when the node processing is successfully completed, that is, after all ESQL has been processed, even if the message flow itself detects an error in a subsequent node that causes the message to be rolled back.

The value that you choose is implemented for the database tables that you have updated. You cannot select a different value for each table.

If you have set Transaction to Commit, the behavior of the message flow and the commitment of database updates can be affected by the use of the PROPAGATE statement.

If you choose to include a PROPAGATE statement in the node's ESQL that generates one or more output message from the node, the processing of the PROPAGATE statement is not considered complete until the entire path that the output message takes has completed. This path might include several other nodes, including one or more output nodes. Only then does the node that issues the PROPAGATE statement receive control back and its ESQL terminate. At that point, a database commit is performed, if appropriate.

If one of the nodes on the propagated path detects an error and throws an exception, the processing of the node in which you have coded the PROPAGATE statement never completes. If the error processing results in a rollback, the message flow and the database update in this node are rolled back. This behavior is consistent with the stated operation of the Commit option, but might not be the behavior that you expect.

Invoking stored procedures:

To invoke a procedure that is stored in a database, use the ESQL CALL statement. The stored procedure must be defined by a CREATE PROCEDURE statement that has a Language clause of DATABASE and an EXTERNAL NAME clause that identifies the name of the procedure in the database and, optionally, the database schema to which it belongs.

When you invoke a stored procedure with the CALL statement, the broker ensures that the ESQL definition and the database definition match:

- The external name of the procedure must match a procedure in the database.
- The number of parameters must be the same.
- The type of each parameter must be the same.
- The direction of each parameter (IN, OUT, INOUT) must be the same.

The following restrictions apply to the use of stored procedures:

- Overloaded procedures are not supported. (An overloaded procedure is one that has the same name as another procedure in the same database schema with a different number of parameters, or parameters with different types.) If the broker detects that a procedure has been overloaded, it raises an exception.
- In an Oracle stored procedure declaration, you are not permitted to constrain CHAR and VARCHAR2 parameters with a length, and NUMBER parameters with a precision or scale, or both. Use %TYPE when you declare CHAR, VARCHAR and NUMBER parameters to provide constraints on a formal parameter.

You can also invoke a database stored procedure or a database user-defined function from a Mapping node. See “Mapping a target element from database stored procedures” on page 585 or “Mapping a target element from database user-defined functions” on page 586.

Creating a stored procedure in ESQL:

When you define an ESQL procedure that corresponds to a database stored procedure, you can specify either a qualified name (where the qualifier is a database schema) or an unqualified name.

To create a stored procedure:

1. Code a statement similar to this example to create an unqualified procedure:

   ```
   CREATE PROCEDURE myProc1(IN p1 CHAR) LANGUAGE DATABASE EXTERNAL NAME "myProc";
   ```

   The EXTERNAL NAME that you specify must match the definition you have created in the database, but you can specify any name you choose for the corresponding ESQL procedure.
2. Code a statement similar to this example to create a qualified procedure:

   ```
   CREATE PROCEDURE myProc2(IN p1 CHAR) LANGUAGE DATABASE EXTERNAL NAME "Schema1.myProc";
   ```
3. Code a statement similar to this example to create a qualified procedure in an Oracle package:

   ```
   CREATE PROCEDURE myProc3(IN p1 CHAR) LANGUAGE DATABASE EXTERNAL
                   NAME "mySchema.myPackage.myProc";
   ```

For examples of stored procedure definitions in the database, see the CREATE PROCEDURE statement.

Calling a stored procedure:

1. Code a statement similar to this example to invoke an unqualified procedure:

   ```
   CALL myProc1('HelloWorld');
   ```

   Because it is not defined explicitly as belonging to any schema, the `myProc1` procedure must exist in the default schema (the name of which is the user name used to connect to the data source) or the command fails.
2. The following example calls the `myProc` procedure in schema `Schema1`.

   ```
   CALL myProc2('HelloWorld');
   ```
3. Code a statement similar to this example to invoke an unqualified procedure with a dynamic schema:

   ```
   DECLARE Schema2 char 'mySchema2';
   CALL myProc1('HelloWorld') IN Database.{'Schema2'};
   ```

   This statement calls the `myProc1` procedure in database `Schema2`, overriding the default “`username`” schema.

Calling a stored procedure that returns two result sets:

To call a stored procedure that takes one input parameter and returns one output parameter and two result sets:

1. Define the procedure with a CREATE PROCEDURE statement that specifies one input parameter, one output parameter, and two result sets:

   ```
   CREATE PROCEDURE myProc1 (IN P1 INT, OUT P2 INT)
     LANGUAGE DATABASE
     DYNAMIC RESULT SETS 2
     EXTERNAL NAME "myschema.myproc1";
   ```
2. To invoke the `myProc1` procedure using a field reference, code:

```
/* using a field reference */
CALL myProc1(InVar1, OutVar2, Environment.ResultSet1[],
             OutputRoot.XMLNS.Test.ResultSet2[]);
```

3. To invoke the myProc1 procedure using a reference variable, code:

```
/* using a reference variable*/
DECLARE cursor REFERENCE TO OutputRoot.XMLNS.Test;

CALL myProc1(InVar1, cursor.OutVar2, cursor.ResultSet1[],
                     cursor.ResultSet2[]);
```

Coding ESQL to handle errors

When you process messages in a message flow, errors can have a number of different causes and the message flow designer must decide how to handle those errors.

Introduction

When you process messages in message flows, errors can have the following causes:

- External causes; for example, the incoming message is syntactically invalid, a database used by the flow has been shut down, or the power supply to the machine on which the broker is running fails.
- Internal causes; for example, an attempt to insert a row into a database table fails because of a constraint check, or a character string that is read from a database cannot be converted to a number because it contains alphabetic characters.

  Internal errors can be caused by programs storing invalid data in the database, or by a flaw in the logic of a flow.

The message flow designer must decide how to handle errors.

Using default error-handling

The simplest strategy for handling ESQL errors is to do nothing, and use the broker's default behavior. The default behavior is to cut short the processing of the failing message, and to proceed to the next message. Input and output nodes provide options to control exactly what happens when processing is cut short.

If the input and output nodes are set to transactional mode, the broker restores the state before the message is processed:

1. The input message that has apparently been taken from the input queue is put back.
2. Any output messages that the flow has apparently written to output queues are discarded.

If the input and output nodes are not set to transactional mode:

1. The input message that was taken from the input queue is not put back.
2. Any output messages that the flow has written to output queues remain on the output queues.

Each of these strategies has its advantages. The transactional model preserves the consistency of data, while the non-transactional model maximizes the continuity of message processing. In the transactional model, the failing input message is put back onto the input queue, and the broker attempts to process it again. The most likely outcome of this scenario is that the message continues to fail until the retry

limit is reached, at which point the message is placed on a dead letter queue. The reason for the failure to process the message is logged to the system event log (Windows) or syslog (UNIX). Therefore, the failing message holds up the processing of subsequent valid messages, and is left unprocessed by the broker.

Most databases operate transactionally so that all changes that are made to database tables are committed if the processing of the message succeeds, or rolled back if it fails, therefore maintaining the integrity of data. An exception to this situation is if the broker itself, or a database, fails (for example, the power to the computers on which they are running is interrupted). In these cases, changes might be committed in some databases, but not in others, or the database changes might be committed but the input and output messages are not committed. If these possibilities concern you, make the flow coordinated and configure the databases that are involved.

Using customized error handling: The following list contains some general tips for creating customized error handlers.

- If you require something better than default error handling, the first step is to use a handler; see DECLARE HANDLER statement. Create one handler per node to intercept all possible exceptions (or all those that you can predict).
- Having intercepted an error, the error handler can use whatever logic is appropriate to handle it. Alternatively, it can use a THROW statement or node to create an exception, which could be handled higher in the flow logic, or even reach the input node, causing the transaction to be rolled back; see "Throwing an exception" on page 382.
- If a node generates an exception that is not caught by the handler, the flow is diverted to the Failure terminal, if one is connected, or handled by default error-handling if no Failure terminal is connected.

  Use Failure terminals to catch unhandled errors. Attach a simple logic flow to the Failure terminal. This logic flow could consist of a database or Compute node that writes a log record to a database (possibly including the message's bit stream), or writes a record to the event log. The flow could also contain an output node that writes the message to a special queue.

  The full exception tree is passed to any node that is connected to a Failure terminal; see "Exception list tree structure" on page 85.
- Your error handlers are responsible for logging each error in an appropriate place, such as the system event log.

For a detailed description of the options that you can use to process errors in a message flow, see "Handling errors in message flows" on page 244. For examples of what you can do, see "Throwing an exception" on page 382 and "Capturing database state" on page 383.

Writing code to detect errors

The following sections assume that the broker detects the error. It is possible, however, for the logic of the flow to detect an error. For example, when coding the flow logic, you could use the following elements:

- IF statements that are inserted specifically to detect situations that should not occur
- The ELSE clause of a case expression or statement to trap routes through the code that should not be possible

As an example of a flow logic-detected error, consider a field that has a range of possible integer values that indicate the type of message. It would not be good

practice to leave to chance what would happen if a message were to arrive in which the field's value did not correspond to any known type of message. One way this situation could occur is if the system is upgraded to support extra types of message, but one part of the system is not upgraded.

Using your own logic to handle input messages that are not valid

Input messages that are syntactically invalid (and input messages that appear to be not valid because of erroneous message format information) are difficult to deal with, because the broker cannot determine the contents of the message. Typically, the best way to deal with these messages is to configure the input node to fully parse and validate the message. However, this configuration applies only to predefined messages; that is, MRM or IDoc.

If the input node is configured in this way, the following results are guaranteed if the input message cannot be parsed successfully:
- The input message never emerges from the node's normal output terminal (it goes to the Failure terminal).
- The input message never enters the main part of the message flow.
- The input message never causes any database updates.
- No messages are written to any output queues.

To deal with a failing message, connect a simple logic flow to the Failure terminal. The only disadvantage to this strategy is that if the normal flow does not require access to all of the message's fields, the forcing of complete parsing of the message affects performance.

Using your own logic to handle database errors

Database errors fall into three categories:
- The database is not working (for example, it is off line).
- The database is working but refuses your request (for example, a lock contention occurs).
- The database is working but it cannot do what you request (for example, read from a non-existent table).

If you require something better than default error handling, the first step is to use a handler (see DECLARE HANDLER statement) to intercept the exception. The handler can determine the nature of the failure from the SQL state that is returned by the database.

A database is not working

If a database is not working at all, and is essential to the processing of messages, there is typically not much that you can do. The handler, having determined the cause, might complete one or more of the following actions:
- Use the RESIGNAL statement to re-throw the original error, therefore allowing the default error handler to take over
- Use a different database
- Write the message to a special output queue

  Be careful if you use an approach similar to this technique; the handler absorbs the exception, therefore all changes to other databases, or writes to queues, are committed.

A database refuses your request

The situation when a lock contention occurs is similar to the “Database not

working" case because the database will have backed out *all* the database changes that you have made for the current message, not just the failing request. Therefore, unless you are sure that this was the only update, default error-handling is typically the best strategy, except possibly logging the error or passing the message to a special queue.

Impossible requests

An impossible request occurs when the database is working, but cannot complete the requested action. This type of error covers a wide variety of problems.

If, as discussed in the previous example, the database does not have a table of the name that the flow expects, default error-handling is typically the best strategy, except possibly logging the error or passing the message to a special queue.

Many other errors might be handled successfully, however. For example, an attempt to insert a row might fail because there is already such a row and the new row would be a duplicate. Or an attempt to update a row might fail because there is no such row (that is, the update action updated zero rows). In these cases, the handler can incorporate whatever logic you think appropriate. It might insert the missing row, or use the existing one (possibly making sure the values in it are suitable).

If you want an update of zero rows to be reported as an error, you must set the Treat warnings as errors property on the node to true, which is not the default setting.

Using your own logic to handle errors in output nodes

Errors that occur in MQOutput nodes report the nature of the error in the SQL state and give additional information in the *SQL native error* variable. Therefore, if something better than default error handling is required, the first step is to use a handler (see DECLARE HANDLER statement) to intercept the exception. Such a handler typically surrounds only a single PROPAGATE statement.

Using your own logic to handle other errors

Besides those errors covered above, a variety of other errors can occur. For example, an arithmetic calculation might overflow, a cast might fail because of the unsuitability of the data, or an access to a message field might fail because of a type constraint. The broker offers two programming strategies for dealing with these types of error.

- The error causes an exception that is either handled or left to roll back the transaction.
- The failure is recorded as a special value that is tested for later.

In the absence of a type constraint, an attempt to access a non-existent message field results in the value null. Null values propagate through expressions, making the result null. Therefore, if an expression, however complex, does not return a null value, you know that all the values that it needed to calculate its result were not null.

Cast expressions can have a default clause. If there is a default clause, casts fail quietly; instead of throwing an exception, they simply return the default value. The default value could be an innocuous number (for example, zero for an integer), or a value that is clearly invalid in the context (for example, -1 for a customer number). Null might be particularly suitable because it is a value that is different

from all others, and it will propagate through expressions without any possibility of the error condition being masked.

Handling errors in other nodes

Exceptions that occur in other nodes (that is, downstream of a PROPAGATE statement) might be caught by handlers in the normal way. Handling such errors intelligently, however, poses a problem: another node was involved in the original error, therefore another node, and not necessarily the originator of the exception, is likely to be involved in handling the error.

To help in these situations, the Database and Compute nodes have four terminals called Out1, Out2, Out3, and Out4. In addition, the syntax of the PROPAGATE statement includes target expression, message source, and control clauses to give more control over these terminals.

Throwing an exception:

If you detect an error or other situation in your message flow in which you want message processing to be ended, you can throw an exception in a message flow in two ways.

- Use the ESQL THROW EXCEPTION statement.

  Include the THROW statement anywhere in the ESQL module for a Compute, Database, or Filter node. Use the options on the statement to code your own data to be inserted into the exception.
- Include a THROW node in your message flow.

  Set the node properties to identify the source and content of the exception.

By using either statement options or node properties, you can specify a message identifier and values that are inserted into the message text to give additional information and identification to users who interpret the exception. You can specify any message in any catalog that is available to the broker. See Using error logging from a user-defined extension for more information.

The situations in which you might want to throw an exception are determined by the behavior of the message flow; decide when you design the message flow where this action might be appropriate. For example, you might want to examine the content of the input message to ensure that it meets criteria that cannot be detected by the input node (which might check that a particular message format is received).

The following example uses the example Invoice message to show how you can use the ESQL THROW statement. To check that the invoice number is within a particular range, throw an exception for any invoice message received that does not fall in the valid range.

```
--Check for invoice number lower than permitted range
IF Body.Invoice.InvoiceNo  < 100000 THEN
   THROW USER EXCEPTION CATALOG 'MyCatalog' MESSAGE 1234 VALUES
   ('Invoice number too low', Body.Invoice.InvoiceNo);

-- Check for invoice number higher than permitted range
ELSEIF Body.InvoiceNo> 500000 THEN
      THROW USER EXCEPTION CATALOG 'MyCatalog' MESSAGE 1235 VALUES
   ('Invoice number too high', Body.Invoice.InvoiceNo);

ELSE DO
  -- invoice number is within permitted range
  -- complete normal processing
ENDIF;
```

Capturing database state:

If an error occurs when the broker accesses an external database, you can either let the broker throw an exception during node processing or use ESQL statements to process the exception within the node itself.

Letting the broker throw an exception during node processing is the default action; ESQL processing in the current node is abandoned. The exception is then propagated backwards through the message flow until an enclosing catch node, or the input node for this message flow, is reached. If the exception reaches the input node, an active transaction is rolled back.

Using ESQL statements to process the exception within the node itself requires an understanding of database return codes and a logical course of action to take when an error occurs. To enable this inline database error processing, you must clear the Filter, Database, or Compute node's Throw Exception On Database Error property. If you clear this property, the node sets the database state indicators SQLCODE, SQLSTATE, SQLNATIVEERROR, and SQLERRORTEXT, with appropriate information from the database manager instead of throwing an exception.

The indicators contain information only when an error (not a warning) occurs, unless you have selected the Treat Warnings As Errors property. If a database operation is successful, or returns success with information, the indicators contain their default success values.

You can use the values contained in these indicators in ESQL statements to make decisions about the action to take. You can access these indicators with the SQLCODE, SQLSTATE, SQLNATIVEERROR, and SQLERRORTEXT functions.

If you are attempting inline error processing, check the state indicators after each database statement is executed to ensure that you catch and assess all errors. When processing the indicators, if you meet an error that you cannot handle inline, you can raise a new exception either to deal with it upstream in a catch node, or to let it through to the input node so that the transaction is rolled back, for which you can use the ESQL THROW statement.

You might want to check for the special case in which a SELECT returns no data. This situation is not considered an error and SQLCODE is not set, therefore you must test explicitly for it; see “Checking returns to SELECT” on page 374.

Using ESQL to access database state indicators

The following ESQL example shows how to use the four database state functions, and how to include the error information that is returned in an exception:

```
DECLARE SQLState1 CHARACTER;
DECLARE SQLErrorText1 CHARACTER;
DECLARE SQLCode1 INTEGER;
DECLARE SQLNativeError1 INTEGER;

-- Make a database insert to a table that does not exist --
INSERT INTO Database.DB2ADMIN.NONEXISTENTTABLE (KEY,QMGR,QNAME)
                              VALUES (45,'REG356','my TESTING 2');

--Retrieve the database return codes --
SET SQLState1 = SQLSTATE;
SET SQLCode1 = SQLCODE;
SET SQLErrorText1 = SQLERRORTEXT;
SET SQLNativeError1 = SQLNATIVEERROR;

--Use the THROW statement to back out the database and issue a user exception--
THROW USER EXCEPTION MESSAGE 2950 VALUES
( 'The SQL State' , SQLState1 , SQLCode1 , SQLNativeError1 ,
SQLErrorText1 );
```

You do not have to throw an exception when you detect a database error; you might prefer to save the error information returned in the local environment tree, and include a Filter node in your message flow that routes the message to error or success subflows according to the values saved.

The following sample program provides another example of ESQL that uses these database functions:

- Airline Reservations

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

Using the SELECT function

The SELECT function is a convenient and powerful tool for accessing fields and transforming data in a message tree.

The following topics show by example how to use the SELECT function to transform a variety of messages. The examples are based on an XML input message that has been parsed in the XMLNS domain. However, the techniques shown in these topics can be applied to any message tree.

- "Transforming a simple message"
- "Transforming a complex message" on page 388
- "Returning a scalar value in a message" on page 390
- "Joining data in a message" on page 392
- "Translating data in a message" on page 393
- "Joining data from messages and database tables" on page 395

Transforming a simple message:

When you code the ESQL for a Compute node, use the SELECT function to transform simple messages.

This topic provides examples of simple message transformation. Review the examples and modify them for your own use. They are all based on the Invoice message as input.

Consider the following ESQL:

```
 SET OutputRoot.XMLNS.Data.Output[] =
    (SELECT R.Quantity, R.Author FROM InputRoot.XMLNS.Invoice.Purchases.Item[] AS R
    );
```

When this ESQL code processes the Invoice message, it produces the following output message:

```
<Data>
  <Output>
     <Quantity>2</Quantity>
     <Author>Neil Bradley</Author>
  </Output>
  <Output>
     <Quantity>1</Quantity>
     <Author>Don Chamberlin</Author>
  </Output>
  <Output>
     <Quantity>1</Quantity>
     <Author>Philip Heller, Simon Roberts</Author>
  </Output>
</Data>
```

Three Output fields are present, one for each Item field, because SELECT creates an item in its result list for each item described by its FROM list. Within each Output field, a Field is created for each field named in the SELECT clause. These fields are in the order in which they are specified within the SELECT clause, not in the order in which they appear in the incoming message.

The R that is introduced by the final AS keyword is known as a correlation name. It is a local variable that represents in turn each of the fields addressed by the FROM clause. The name chosen has no significance. In summary, this simple transform does two things:

1. It discards unwanted fields.
2. It guarantees the order of the fields.

You can perform the same transform with a procedural algorithm:

```
DECLARE i INTEGER 1;
DECLARE count INTEGER CARDINALITY(InputRoot.XMLNS.Invoice.Purchases.Item[]);

WHILE (i <= count)
   SET OutputRoot.XMLNS.Data.Output[i].Quantity = InputRoot.XMLNS.Invoice.Purchases.Item[i].Quantity;
   SET OutputRoot.XMLNS.Data.Output[i].Author   = InputRoot.XMLNS.Invoice.Purchases.Item[i].Author;
   SET i = i+1;
END WHILE;
```

These examples show that the SELECT version of the transform is much more concise. It also executes faster.

The following example shows a more advanced transformation:

```
SET OutputRoot.XMLNS.Data.Output[] =
    (SELECT R.Quantity AS Book.Quantity,
            R.Author   AS Book.Author
            FROM InputRoot.XMLNS.Invoice.Purchases.Item[] AS R
    );
```

In this transform, an AS clause is associated with each item in the SELECT clause. This clause gives each field in the result an explicit name rather than a field name that is inherited from the input. These names can be paths (that is, a dot-separated list of names), as shown in the example. The structure of the output message structure can be different from the input message. Using the same Invoice message, the result is:

```
<Data>
  <Output>
    <Book>
      <Quantity>2</Quantity>
      <Author>Neil Bradley</Author>
    </Book>
  </Output>
  <Output>
    <Book>
      <Quantity>1</Quantity>
      <Author>Don Chamberlin</Author>
    </Book>
  </Output>
  <Output>
    <Book>
      <Quantity>1</Quantity>
      <Author>Philip Heller, Simon Roberts</Author>
    </Book>
  </Output>
</Data>
```

The expressions in the SELECT clause can be of any complexity and there are no special restrictions. They can include operators, functions, and literals, and they can refer to variables or fields that are not related to the correlation name. The following example shows more complex expressions:

```
SET OutputRoot.XMLNS.Data.Output[] =
    (SELECT 'Start'                          AS Header,
            'Number of books:' || R.Quantity AS Book.Quantity,
            R.Author || ':Name and Surname'  AS Book.Author,
            'End'                            AS Trailer
            FROM InputRoot.XMLNS.Invoice.Purchases.Item[] AS R
    );
```

Using the same Invoice message, the result in this case is:

```
<Data>
 <Output>
  <Header>Start</Header>
  <Book>
   <Quantity>Number of books:2</Quantity>
   <Author>Neil Bradley:Name and Surname</Author>
  </Book>
  <Trailer>End</Trailer>
 </Output>
 <Output>
  <Header>Start</Header>
  <Book>
   <Quantity>Number of books:1</Quantity>
   <Author>Don Chamberlin:Name and Surname</Author>
  </Book>
  <Trailer>End</Trailer>
 </Output>
 <Output>
  <Header>Start</Header>
  <Book>
   <Quantity>Number of books:1</Quantity>
   <Author>Philip Heller, Simon Roberts:Name and Surname</Author>
  </Book>
  <Trailer>End</Trailer>
 </Output>
</Data>
```

As shown above, the AS clauses of the SELECT clause contain a path that describes the full name of the field that is to be created in the result. These paths can also specify (as is normal for paths) the type of field that is to be created. The following example transform specifies the field types. In this case, XML tagged data is transformed to XML attributes:

```
SET OutputRoot.XMLNS.Data.Output[] =
    (SELECT R.Quantity.* AS Book.(XML.Attribute)Quantity,
            R.Author.*   AS Book.(XML.Attribute)Author
            FROM InputRoot.XMLNS.Invoice.Purchases.Item[] AS R
    );
```

Using the same Invoice message, the result is:

```
<Data>
 <Output>
  <Book Quantity="2" Author="Neil Bradley"/>
 </Output>
 <Output>
  <Book Quantity="1" Author="Don Chamberlin"/>
 </Output>
 <Output>
  <Book Quantity="1" Author="Philip Heller, Simon Roberts"/>
 </Output>
</Data>
```

Finally, you can use a WHERE clause to eliminate some of the results. In the following example a WHERE clause is used to remove results in which a specific criterion is met. An entire result is either included or excluded:

```
SET OutputRoot.XMLNS.Data.Output[] =
    (SELECT R.Quantity AS Book.Quantity,
            R.Author   AS Book.Author
            FROM InputRoot.XMLNS.Invoice.Purchases.Item[] AS R
            WHERE R.Quantity = 2
    );
```

Using the same input message, the result is:

```
<Data>
 <Output>
  <Book>
   <Quantity>2</Quantity>
   <Author>Neil Bradley</Author>
  </Book>
 </Output>
</Data>
```

Transforming a complex message:

When you code the ESQL for a Compute node, use the SELECT function for complex message transformation.

This topic provides examples of complex message transformation. Review the examples and modify them for your own use. They are all based on the Invoice message as input.

In this example, Invoice contains a variable number of Items. The transform is shown in the following example:

```
SET OutputRoot.XMLNS.Data.Statement[] =
    (SELECT I.Customer.Title                                    AS Customer.Title,
            I.Customer.FirstName || ' ' || I.Customer.LastName  AS Customer.Name,
            COALESCE(I.Customer.PhoneHome,'')                   AS Customer.Phone,
            (SELECT II.Title                          AS Desc,
                    CAST(II.UnitPrice AS FLOAT) * 1.6 AS Cost,
                    II.Quantity                       AS Qty
             FROM I.Purchases.Item[] AS II
             WHERE  II.UnitPrice> 0.0             )     AS Purchases.Article[],
            (SELECT SUM( CAST(II.UnitPrice AS FLOAT) *
                    CAST(II.Quantity  AS FLOAT) *
                    1.6                           )
             FROM I.Purchases.Item[] AS II      )       AS Amount,
            'Dollars'                                   AS Amount.(XML.Attribute)Currency

            FROM InputRoot.XMLNS.Invoice[] AS I
            WHERE I.Customer.LastName <> 'Brown'
    );
```

The output message that is generated is:

```
<Data>
 <Statement>
  <Customer>
   <Title>Mr</Title>
   <Name>Andrew Smith</Name>
   <Phone>01962818000</Phone>
  </Customer>
  <Purchases>
   <Article>
    <Desc Category="Computer" Form="Paperback" Edition="2">The XML Companion</Desc>
    <Cost>4.472E+1</Cost>
    <Qty>2</Qty>
   </Article>
   <Article>
    <Desc Category="Computer" Form="Paperback" Edition="2">
          A Complete Guide to DB2 Universal Database</Desc>
    <Cost>6.872E+1</Cost>
    <Qty>1</Qty>
   </Article>
   <Article>
    <Desc Category="Computer" Form="Hardcover" Edition="0">JAVA 2 Developers Handbook</Desc>
    <Cost>9.5984E+1</Cost>
    <Qty>1</Qty>
   </Article>
  </Purchases>
  <Amount Currency="Dollars">2.54144E+2</Amount>
 </Statement>
</Data>
```

This transform has nested SELECT clauses. The outer statement operates on the list of Invoices. The inner statement operates on the list of Items. The AS clause that is associated with the inner SELECT clause expects an array:

```
(SELECT II.Title                          AS Desc,
        CAST(II.UnitPrice AS FLOAT) * 1.6 AS Cost,
        II.Quantity                       AS Qty
 FROM I.Purchases.Item[] AS II
 WHERE  II.UnitPrice> 0.0             )
-- Note the use of [] in the next expression
   AS Purchases.Article[],
```

This statement tells the outer SELECT clause to expect a variable number of Items in each result. Each SELECT clause has its own correlation name: I for the outer SELECT clause and II for the inner one. Each SELECT clause typically uses its own correlation name, but the FROM clause in the inner SELECT clause refers to the correlation name of the outer SELECT clause:

```
(SELECT II.Title                          AS Desc,
        CAST(II.UnitPrice AS FLOAT) * 1.6 AS Cost,
        II.Quantity                       AS Qty
-- Note the use of I.Purchases.Item in the next expression
 FROM I.Purchases.Item[] AS II
 WHERE  II.UnitPrice> 0.0             ) AS Purchases.Article[],
```

This statement tells the inner SELECT clause to work with the current Invoice's Items. Both SELECT clauses contain WHERE clauses. The outer one uses one criterion to discard certain Customers, and the inner one uses a different criterion to discard certain Items. The example also shows the use of COALESCE to prevent missing input fields from causing the corresponding output field to be missing. Finally, it also uses the column function SUM to add together the value of all Items in each Invoice. Column functions are discussed in "Referencing columns in a database" on page 367.

When the fields Desc are created, the whole of the input Title field is copied: the XML attributes and the field value. If you do not want these attributes in the output message, use the FIELDVALUE function to discard them; for example, code the following ESQL:

```
SET OutputRoot.XMLNS.Data.Statement[] =
    (SELECT I.Customer.Title                                AS Customer.Title,
            I.Customer.FirstName || ' ' || I.Customer.LastName AS Customer.Name,
            COALESCE(I.Customer.PhoneHome,'')               AS Customer.Phone,
            (SELECT FIELDVALUE(II.Title)                    AS Desc,
                    CAST(II.UnitPrice AS FLOAT) * 1.6 AS Cost,
                    II.Quantity                       AS Qty
             FROM I.Purchases.Item[] AS II
             WHERE  II.UnitPrice> 0.0             )      AS Purchases.Article[],
            (SELECT SUM( CAST(II.UnitPrice AS FLOAT) *
                    CAST(II.Quantity  AS FLOAT) *
                    1.6                           )
             FROM I.Purchases.Item[] AS II        ) AS Amount,
            'Dollars'                               AS Amount.(XML.Attribute)Currency

    FROM InputRoot.XMLNS.Invoice[] AS I
    WHERE I.Customer.LastName <> 'Brown'
    );
```

That code generates the following output message:

```
<Data>
 <Statement>
  <Customer>
   <Title>Mr</Title>
   <Name>Andrew Smith</Name>
   <Phone>01962818000</Phone>
  </Customer>
  <Purchases>
   <Article>
    <Desc>The XML Companion</Desc>
    <Cost>4.472E+1</Cost>
    <Qty>2</Qty>
   </Article>
   <Article>
    <Desc>A Complete Guide to DB2 Universal Database</Desc>
    <Cost>6.872E+1</Cost>
    <Qty>1</Qty>
   </Article>
   <Article>
    <Desc>JAVA 2 Developers Handbook</Desc>
    <Cost>9.5984E+1</Cost>
    <Qty>1</Qty>
   </Article>
  </Purchases>
  <Amount Currency="Dollars">2.54144E+2</Amount>
 </Statement>
</Data>
```

Returning a scalar value in a message:

Use a SELECT statement to return a scalar value by including both the THE and ITEM keywords.

For example:

```
1 + THE(SELECT ITEM T.a FROM Body.Test.A[] AS T WHERE T.b = '123')
```

Use of the ITEM keyword:

The following example shows the use of the ITEM keyword to select one item and create a single value.

```
SET OutputRoot.MQMD = InputRoot.MQMD;

SET OutputRoot.XMLNS.Test.Result[] =
   (SELECT ITEM T.UnitPrice FROM InputBody.Invoice.Purchases.Item[] AS T);
```

When the Invoice message is received as input, the ESQL shown generates the following output message:

```
<Test>
  <Result>27.95</Result>
  <Result>42.95</Result>
  <Result>59.99</Result>
</Test>
```

When the ITEM keyword is specified, the output message includes a list of scalar values. Compare this message to the one that is produced if the ITEM keyword is omitted, in which a list of fields (name-value pairs) is generated:

```
<Test>
  <Result>
    <UnitPrice>27.95</UnitPrice>
  </Result>
  <Result>
    <UnitPrice>42.95</UnitPrice>
  </Result>
  <Result>
    <UnitPrice>59.99</UnitPrice>
  </Result>
</Test>
```

Effects of the THE keyword:

The THE keyword converts a list containing one item to the item itself.

The two previous examples both specified a list as the source of the SELECT in the FROM clause (the field reference has [] at the end to indicate an array), so typically the SELECT function generates a list of results. Because of this behavior, you must specify a list as the target of the assignment (thus the "Result[]" as the target of the assignment). However, you often know that the WHERE clause that you specify as part of the SELECT returns TRUE for only one item in the list. In this case use the THE keyword.

The following example shows the effect of using the THE keyword:

```
SET OutputRoot.MQMD = InputRoot.MQMD;

SET OutputRoot.XMLNS.Test.Result =
    THE (SELECT T.Publisher, T.Author FROM InputBody.Invoice.Purchases.Item[]
         AS T WHERE T.UnitPrice = 42.95);
```

The THE keyword means that the target of the assignment becomes `OutputRoot.XMLNS.Test.Result` (the "[]" is not permitted). Its use generates the following output message:

```
<Test>
  <Result>
    <Publisher>Morgan Kaufmann Publishers</Publisher>
    <Author>Don Chamberlin</Author>
  </Result>
</Test>
```

Selecting from a list of scalars:

Consider the following sample input message:

```
<Test>
 <A>1</A>
 <A>2</A>
 <A>3</A>
 <A>4</A>
 <A>5</A>
</Test>
```

If you code the following ESQL statements to process this message:

```
SET OutputRoot.XMLNS.Test.A[] =
  (SELECT ITEM A from InputBody.Test.A[]
   WHERE CAST(A AS INTEGER) BETWEEN 2 AND 4);
```

the following output message is generated:

```
      <A>2</A>
      <A>3</A>
      <A>4</A>
```

Joining data in a message:

The FROM clause of a SELECT function is not restricted to having one item. Specifying multiple items in the FROM clause produces the typical Cartesian product joining effect, in which the result includes an item for all combinations of items in the two lists.

Using the FROM clause in this way produces the same joining effect as standard SQL.

The Invoice message includes a set of customer details, payment details, and details of the purchases that the customer makes. Code the following ESQL to process the input Invoice message:

```
SET OutputRoot.XMLNS.Items.Item[] =
   (SELECT D.LastName, D.Billing,
           P.UnitPrice, P.Quantity
    FROM InputBody.Invoice.Customer[] AS D,
         InputBody.Invoice.Purchases.Item[] AS P);
```

The following output message is generated:

```
<Items>
 <Item>
  <LastName>Smith</LastName>
  <Billing>
   <Address>14 High Street</Address>
   <Address>Hursley Village</Address>
   <Address>Hampshire</Address>
   <PostCode>SO213JR</PostCode>
  </Billing>
  <UnitPrice>27.95</UnitPrice>
  <Quantity>2</Quantity>
 </Item>
 <Item>
  <LastName>Smith</LastName>
  <Billing>
   <Address>14 High Street</Address>
   <Address>Hursley Village</Address>
   <Address>Hampshire</Address>
   <PostCode>SO213JR</PostCode>
  </Billing>
  <UnitPrice>42.95</UnitPrice>
  <Quantity>1</Quantity>
 </Item>
 <Item>
  <LastName>Smith</LastName>
  <Billing>
   <Address>14 High Street</Address>
   <Address>Hursley Village</Address>
   <Address>Hampshire</Address>
   <PostCode>SO213JR</PostCode>
  </Billing>
  <UnitPrice>59.99</UnitPrice>
  <Quantity>1</Quantity>
 </Item>
</Items>
```

Three results are produced, giving the number of descriptions in the first list (one) multiplied by the number of prices in the second (three). The results systematically work through all the combinations of the two lists. You can see this by looking at the *LastName* and *UnitPrice* fields selected from each result:

```
LastName Smith   UnitPrice 27.95
LastName Smith   UnitPrice 42.95
LastName Smith   UnitPrice 59.99
```

You can join data that occurs in a list and a non-list, or in two non-lists, and so on. For example:

```
OutputRoot.XMLNS.Test.Result1[] =
  (SELECT ... FROM InputBody.Test.A[], InputBody.Test.b);
OutputRoot.XMLNS.Test.Result1 =
  (SELECT ... FROM InputBody.Test.A, InputBody.Test.b);
```

The location of the [] in each case is significant. Any number of items can be specified in the FROM clause, not just one or two. If any of the items specify [] to indicate a list of items, the SELECT function returns a list of results (the list might contain only one item, but the SELECT function can return a list of items).

The target of the assignment must specify a list (so must end in []), or you must use the THE function if you know that the WHERE clause guarantees that only one combination is matched.

Translating data in a message:

You can translate data from one form to another.

A typical example of the requirement to translate data is if the items are known in one message by names, and in another message by numbers. For example:

```
Type Name          Type Code

Confectionary      2000
Newspapers         3000
Hardware           4000
```

Consider the following input message:

```
<Data>
  <Items>
    <Item>
      <Cat>1000</Cat>
      <Description>Milk Chocolate Bar</Description>
      <Type>Confectionary</Type>
    </Item>
    <Item>
      <Cat>1001</Cat>
      <Description>Daily Newspaper</Description>
      <Type>NewsPapers</Type>
    </Item>
    <Item>
      <Cat>1002</Cat>
      <Description>Kitchen Sink</Description>
      <Type>Hardware</Type>
    </Item>
  </Items>
  <TranslateTable>
    <Translate>
      <Name>Confectionary</Name>
      <Number>2000</Number>
    </Translate>
    <Translate>
      <Name>NewsPapers</Name>
      <Number>3000</Number>
    </Translate>
    <Translate>
      <Name>Hardware</Name>
      <Number>4000</Number>
    </Translate>
  </TranslateTable>
</Data>
```

This message has two sections; the first section is a list of items in which each item has a catalogue number and a type; the second section is a table for translating between descriptive type names and numeric type codes. Include a Compute node with the following transform:

```
SET OutputRoot.XMLNS.Result.Items.Item[] =
   (SELECT M.Cat, M.Description, T.Number As Type
     FROM
       InputRoot.XMLNS.Data.Items.Item[]               As M,
       InputRoot.XMLNS.Data.TranslateTable.Translate[] As T
     WHERE M.Type = T.Name
   );
```

The following output message is generated:

```
<Result>
  <Items>
    <Item>
      <Cat>1000</Cat>
      <Description>Milk Chocolate Bar</Description>
      <Type>2000</Type>
    </Item>
    <Item>
      <Cat>1001</Cat>
      <Description>Daily Newspaper</Description>
      <Type>3000</Type>
    </Item>
    <Item>
      <Cat>1002</Cat>
      <Description>Kitchen Sink</Description>
      <Type>4000</Type>
    </Item>
  </Items>
</Result>
```

In the result, each type name has been converted to its corresponding code. In this example, both the data and the translate table were in the same message tree, although this is not a requirement. For example, the translate table could be coded in a database, or might have been set up in LocalEnvironment by a previous Compute node.

Joining data from messages and database tables:

You can use SELECT functions that interact with both message data and databases.

You can also nest a SELECT function that interacts with one type of data within a SELECT clause that interacts with the other type.

Consider the following input message, which contains invoice information for two customers:

```
<Data>
  <Invoice>
    <CustomerNumber>1234</CustomerNumber>
    <Item>
      <PartNumber>1</PartNumber>
      <Quantity>9876</Quantity>
    </Item>
    <Item>
      <PartNumber>2</PartNumber>
      <Quantity>8765</Quantity>
    </Item>
</Invoice>
  <Invoice>
    <CustomerNumber>2345</CustomerNumber>
    <Item>
      <PartNumber>2</PartNumber>
      <Quantity>7654</Quantity>
    </Item>
    <Item>
      <PartNumber>1</PartNumber>
    <Quantity>6543</Quantity>
    </Item>
</Invoice>
</Data>
```

Consider the following database tables, Prices and Addresses, and their contents:

```
PARTNO      PRICE
----------- ----------------------
          1            +2.50000E+001
          2            +6.50000E+00


PARTNO      STREET               CITY             COUNTRY
------      ------------------   --------------   -------
1234        22 Railway Cuttings  East Cheam       England
2345        The Warren           Watership Down   England
```

If you code the following ESQL transform:

```
-- Create a valid output message
SET OutputRoot.MQMD = InputRoot.MQMD;

-- Select suitable invoices
SET OutputRoot.XMLNS.Data.Statement[] =
   (SELECT I.CustomerNumber                          AS Customer.Number,
           A.Street                                  AS Customer.Street,
           A.City                                    AS Customer.Town,
           A.Country                                 AS Customer.Country,

        -- Select suitable items
          (SELECT II.PartNumber AS PartNumber,
                  II.Quantity   AS Quantity,
                  PI.Price      AS Price
           FROM Database.db2admin.Prices AS PI,
               I.Item[]                 AS II
           WHERE II.PartNumber = PI.PartNo     )    AS Purchases.Item[]

    FROM Database.db2admin.Addresses  AS A,
         InputRoot.XMLNS.Data.Invoice[] AS I

    WHERE I.CustomerNumber = A.PartNo
    );
```

the following output message is generated. The input message is augmented with the price and address information from the database table:

```
<Data>
  <Statement>
    <Customer>
      <Number>1234</Number>
      <Street>22 Railway Cuttings</Street>
      <Town>East Cheam</Town>
      <Country>England</Country>
    </Customer>
    <Purchases>
      <Item>
        <PartNumber>1</PartNumber>
        <Quantity>9876</Quantity>
        <Price>2.5E+1</Price>
      </Item>
      <Item>
        <PartNumber>2</PartNumber>
        <Quantity>8765</Quantity>
        <Price>6.5E+1</Price>
      </Item>
    </Purchases>
  </Statement>
  <Statement>
    <Customer>
      <Number>2345</Number>
      <Street>The Warren</Street>
      <Town>Watership Down</Town>
      <Country>England</Country>
    </Customer>
    <Purchases>
      <Item>
        <PartNumber>1</PartNumber>
        <Quantity>6543</Quantity>
        <Price>2.5E+1</Price></Item>
      <Item>
        <PartNumber>2</PartNumber>
        <Quantity>7654</Quantity>
        <Price>6.5E+1</Price>
      </Item>
    </Purchases>
  </Statement>
</Data>
```

You can nest the database SELECT clause within the message SELECT clause. In most cases, the code is not as efficient as the previous example, but you might find that it is better if the messages are small and the database tables are large.

```
-- Create a valid output message
SET OutputRoot.MQMD = InputRoot.MQMD;

-- Select suitable invoices
SET OutputRoot.XMLNS.Data.Statement[] =
    (SELECT I.CustomerNumber                  AS Customer.Number,

        -- Look up the address
        THE ( SELECT
                  A.Street,
                  A.City    AS Town,
                  A.Country
                FROM Database.db2admin.Addresses AS A
                WHERE A.PartNo = I.CustomerNumber
            )                               AS Customer,

        -- Select suitable items
        (SELECT
            II.PartNumber AS PartNumber,
            II.Quantity   AS Quantity,

            -- Look up the price
            THE (SELECT ITEM P.Price
              FROM Database.db2admin.Prices AS P
              WHERE P.PartNo = II.PartNumber
            )             AS Price

          FROM I.Item[] AS II           ) AS Purchases.Item[]

    FROM InputRoot.XMLNS.Data.Invoice[] AS I
    );
```

Manipulating messages in the XML domains

The following topics contain instructions on manipulating messages in the XMLNSC, XMLNS, and XML domains.

- "Working with XML messages"
- "Manipulating messages in the XMLNSC domain" on page 408
- "Manipulating messages in the XMLNS domain" on page 421
- "Manipulating messages in the XML domain" on page 431

For information about dealing with MRM XML messages, see "Manipulating messages in the MRM domain" on page 431.

Working with XML messages:

The following topics provide information about typical tasks for processing XML messages.

Some of this information is available publicly in Web pages and online tutorials. If you are new to XML, you will find it useful to also read about the XML standard.

- "Constructing an XML message tree" on page 399
- "Working with namespaces" on page 399
- "Working with binary data" on page 400
- "XMLNSC: Working with CData" on page 402
- "XMLNSC: Working with XML messages and bit streams" on page 403
- "Working with large XML messages" on page 404

For details about XML Schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) Web site.

Constructing an XML message tree:

When constructing an XML message tree, consider the order of fields in the tree.

Order of fields in the message tree

When you create an XML output message in a Compute node, the order of your lines of ESQL code is important, because the message elements are created in the order that you code them.

Consider the following XML message:

```
<Order>
  <ItemNo>1</ItemNo>
  <Quantity>2</Quantity>
</Order>
```

If you want to add a DocType Declaration to this, insert the DocType Declaration before you copy the input message to the output message.

For example:

```
SET OutputRoot.XMLNS.(XML.XmlDecl) = '';
SET OutputRoot.XMLNS.(XML.XmlDecl).(XML.Version) = '1.0';
SET OutputRoot.XMLNS.(XML.DocTypeDecl)Order ='';
SET OutputRoot.XMLNS.(XML.DocTypeDecl).(XML.SystemId)   = 'NewDtdName.dtd';
SET OutputRoot = InputRoot;
  -- more ESQL --
```

If you put the last statement to copy the input message before the XML-specific statements, the following XML is generated for the output message.

```
<Order>
  <ItemNo>1</ItemNo>
  <Quantity>2</Quantity>
</Order>
<?xml version="1.0"?>
```

This is not well-formed XML and causes an error when it is written from the message tree to a bit stream in the output node.

Setting the field type

If you copy a message tree from input to output without changing the domain, most of the syntax elements will be created by the parser (XMLNSC or XMLNS) and their field types will be correct. However, if you construct your message tree from a database query, or from another parser's message tree, you must ensure that you identify each syntax element correctly by using its field type. You can find full details of the field type constants used by XMLNSC and XMLNS in the following topics:

- "XMLNSC: Using field types" on page 107
- "XML constructs" on page 1567

Working with namespaces:

The following example shows how to use ESQL to work with namespaces.

Namespace constants are declared at the start of the main module so that you can use prefixes in the ESQL statements instead of the full URIs of the namespaces. The namespace constants affect only the ESQL; they do not control the prefixes that are generated in the output message. The prefixes in the generated output message are controlled by namespace declarations in the message tree. You can include namespace declarations in the tree using the XML.NamespaceDecl field type. These elements are then used to generate namespace declarations in the output message.

When the output message is generated, if the parser encounters a namespace for which it has no corresponding namespace declaration, a prefix is automatically generated using prefixes of the form NSn where n is a positive integer.

```
CREATE COMPUTE MODULE xmlns_doc_flow_Compute
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
CALL CopyMessageHeaders();
-- Declaration of namespace constants --
These are only used by ESQL

DECLARE sp1 NAMESPACE 'http://www.ibm.com/space1';
DECLARE sp2 NAMESPACE 'http://www.ibm.com/space2';
DECLARE sp3 NAMESPACE 'http://www.ibm.com/space3';

-- Namespace declaration for prefix 'space1'
SET OutputRoot.XMLNS.message.(XML.NamespaceDecl)xmlns:space1 = 'http://www.ibm.com/space1';
SET OutputRoot.XMLNS.message.sp1:data1 = 'Hello!';

-- Default namespace declaration ( empty prefix )
SET OutputRoot.XMLNS.message.sp2:data2.(XML.NamespaceDecl)xmlns = 'http://www.ibm.com/space2';
SET OutputRoot.XMLNS.message.sp2:data2.sp2:subData1 = 'Hola!';
SET OutputRoot.XMLNS.message.sp2:data2.sp2:subData2 = 'Guten Tag!';
SET OutputRoot.XMLNS.message.sp3:data3 = 'Bonjour!';
RETURN TRUE;
END;

CREATE PROCEDURE CopyMessageHeaders()
BEGIN
DECLARE I INTEGER 1;
DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
WHILE I < J DO SET OutputRoot.*[I] = InputRoot.*[I];
  SET I = I + 1;
END WHILE;
END;
END MODULE;
```

When this ESQL is processed, the following output message is generated:

```
<message xmlns:space1-"http://www.ibm.com/space1">
 <space1:data1>Hello!</space1:data1>
 <data2 xmlns="http://www.ibm.com/space2">
  <subData1>Hola!</subData1>
  <subData2>Guten Tag!</subData2>
 </data2>
 <NS1:data3 xmlns="http://www.ibm.com/space3>Bonjour!</NS1:data3>
</message>
```

Working with binary data:

If you need to include binary data or non-valid characters in your XML documents, the safest method is to encode the data as a binary string.

Binary encodings for XML

There are two methods of encoding binary data in an XML document.

```
hexBinary: <nonXMLChars>000102030405060708B0C0E0F</nonXMLChars>

base64Binary: <nonXMLChars>AAECAwQFBgcICwwODw==</nonXMLChars>
```

The base64Binary encoding makes better use of the available XML characters, and on average a base64-encoded binary field is 2/3 the size of its hexBinary equivalent. Base64Binary is widely used by the MIME format and by various XML-based standards.

You might prefer to use the simpler hexBinary encoding if you are sending data to applications that cannot decode base64 data, and if the increase in message size is not important.

Parsing binary data

The most straightforward way to parse any binary data is to use the XMLNSC parser with a message set:

1. Locate or construct an XML Schema that describes your input XML.
2. Import the XML Schema to create a message definition file.
3. In your message flow, set the node properties as follows:
   - On the Default page, set *Message Domain* to `XMLNSC` and *Message Set* to the name of your message set.
   - On the Validation page, set *Validation* to `Content and Value`.
   - In the XMLNSC properties, select the check box option *Build tree using XML Schema types*.

The XMLNSC parser automatically decodes your hexBinary or base64Binary data, being guided by the simple type of the element or attribute that contains the binary data. The message tree will contain the decoded BLOB value.

If you are using the XMLNS domain, you must parse the binary data as a string. It appears in the message tree as a CHARACTER value. If the data is encoded as hexBinary, you can use the ESQL CAST function to convert it to a BLOB value. If the data is encoded as base64Binary, the easiest approach is to call a static Java method from ESQL to decode the base64 data into a BLOB value.

Generating binary data

You can generate binary data in your output XML in either hexBinary or base64Binary encoding.

For hexBinary, use the ESQL CAST statement to convert your BLOB data to a hexBinary string.

For base64Binary, you have two options:

- Call a static Java method to encode your BLOB data as base64.
- Use the XMLNSC parser, and modify the type field on the syntax element, as shown in this example:

```
-- ESQL code to generate base64-encoded binary data
DECLARE myBLOB BLOB;
-- Populate myBLOB with your binary data
```

```
CREATE LASTCHILD OF OutputRoot.XMLNSC.message
       TYPE BITOR(XMLNSC.Attribute, XMLNSC.base64Binary)
       NAME myBase64Element
       VALUE myBLOB;
```

Note that setting the field type to XMLNSC.base64Binary does not change the logical value in the message tree. In your message flow, it is still a BLOB, and if you ask for its string representation, it is reported as a hexBinary string. However, when the message tree is converted to a bit stream (in an output node, or by a call to ASBITSTREAM) the base64 conversion is performed automatically, and the output XML contains the correct base64 string.

XMLNSC: Working with CData:

A CData section can be used to embed an XML document within another XML document.

What is a CData section?

An XML element can contain text content:

```
<element>text content</element>
```

However, some characters cannot appear in that content. In particular, '<' and '&' both have special meaning to an XML parser. If they are included in the text content of an element, they change the meaning of the XML document.

For example, this is a badly formed XML document:

```
<element><text><content></element>
```

There are two ways to make the XML well-formed:

1. Use character entities:

   ```
   <element>&lt;text&gt;&lt;content&gt;</element>
   ```

2. Use a CData section:

   ```
   <element><![CDATA[<text><content>]]></element>
   ```

What can you use a CData section for?

In a CData section, you can include XML markup in the value of an element. However, non-valid XML characters cannot be included. Binary data also cannot be included in a CData section.

The most common use for CData is to embed one XML document within another. For example:

```
<outer>
    <embedXML>
        <![CDATA[<innerMsg></innerMsg>]]>
    </embedXML>
</outer>
```

You can even embed a badly-formed XML document in this way, because the XML parser does not attempt to parse the content of a CData section.

```
<outer>
    <embedXML>
        <![CDATA[<badXML></wrongClosingTag>]]>
    </embedXML>
</outer>
```

The following items are not valid within a CData section:

- Non-valid XML characters (see http://www.w3.org/TR/2006/REC-xml-20060816/#charsets)
- The text string ']]>' (because this terminates the CData section)

Because of these restrictions, do not use a CData section to include arbitrary text in your XML document, and do not try to use a CData section to hold binary data (unless it is encoded as hexBinary or base64Binary).

How do you add a CData section to an output XML message?

Consider the following input message :

```
<TestCase>
   <Folder>
      <Field1>Value1</Field1>
      <Field2>Value2</Field2>
      <Field3>Value3</Field3>
   </Folder>
</TestCase>
```

The following ESQL shows how to serialize a whole message:

```
DECLARE wholeMsgBlob BLOB
  ASBITSTREAM(InputRoot.XMLNSC,
              InputRoot.Properties.Encoding,
              InputRoot.Properties.CodedCharSetId );
 DECLARE wholeMsgChar CHAR
  CAST(wholeMsgBlob AS CHAR CCSID InputRoot.Properties.CodedCharSetId);
SET OutputRoot.XMLNSC.Output.(XMLNSC.CDataField)Field1 = wholeMsgChar;
```

This example serializes the InputRoot.XMLNSC.TestCase.Folder portion of the message tree.

If the output message tree were examined before an MQOutput node, this would show :

```
(0x01000010):XML        = (
  (0x01000000):Output = (
    (0x01000000):Field1 = (
      (0x02000001): = '<TestCase><Folder><Field1>Value1</Field1><Field2>Value2</Field2>
                      <Field3>Value3</Field3></Folder><Folder2><Field1>Value1</Field1>
                      <Field2>Value2</Field2><Field3>Value3</Field3><Folder2></TestCase>'
       )
       )
   )
```

As can be seen, each CData section contains a single scalar value that is the character representation of the portion of the XML message that is required.

This tree produces the following XML output message :

```
<Output>
 <Field1><![CDATA[<TestCase><Folder><Field1>Value 1</Field1>
                                   <Field2>Value 2</Field2>
                                   <Field3>Value 3</Field3></Folder>
                           <Folder2><Field1>Value 1</Field1>
                                    <Field2>Value 2</Field2>
                                    <Field3>Value 3</Field3></Folder2>
                 </TestCase>]]</Field1>
 </Output>
```

XMLNSC: Working with XML messages and bit streams:

Use the ASBITSTREAM function and the CREATE statement to manage XML message content.

The ASBITSTREAM function

If you code the ASBITSTREAM function with the parser mode option set to RootBitStream to parse a message tree to a bit stream, the result is an XML document that is built from the children of the target element in the normal way. This algorithm is identical to that used to generate the normal output bit stream. Because the target element is not included in the output bit stream, you must ensure that the children of the element follow the constraints for an XML document.

One constraint is that there must be only one body element in the message. You can use a well-formed bit stream obtained in this way to re-create the original tree using a CREATE statement with a PARSE clause.

If you code the ASBITSTREAM function with the parser mode option set to FolderBitStream to parse a message tree to a bit stream, the generated bit stream is an XML document built from the target element and its children. Any DocTypeDecl or XmlDecl elements are ignored, and the target element itself is included in the generated bit stream.

The advantage of this mode is that the target element becomes the body element of the document, and that body element can have multiple elements nested within it. Use this mode to obtain a bit stream description of arbitrary sub-trees owned by an XML parser. You can use bit streams obtained in this way to re-create the original tree using a CREATE statement with a PARSE clause, and a mode of FolderBitStream.

For further information about the ASBITSTREAM function, and some examples of its use, see ASBITSTREAM function.

The CREATE statement with a PARSE clause

If you code a CREATE statement with a PARSE clause with the parser mode option set to RootBitStream to parse a bit stream to a message tree, the expected bit stream is a normal XML document. A field in the tree is created for each field in the document. This algorithm is identical to that used when parsing a bit stream from an input node. In particular, an element named 'XML', 'XMLNS', or 'XMLNSC' is created as the root element of the tree, and all the content in the message is created as children of that root.

If you code a CREATE statement with a PARSE clause with the parser mode option set to FolderBitStream to parse a bit stream to a message tree, the expected bit stream is a normal XML document. Any content outside the body element (such as an XML declaration or doctype) is discarded. The first element created during the parse corresponds to the body of the XML document, and from there the parse proceeds as normal.

For further information about the CREATE statement, and examples of its use, see CREATE statement.

Working with large XML messages:

When an input bit stream is parsed and a logical tree is created, the tree representation of an XML message is typically bigger, and in some cases much bigger, than the corresponding bit stream.

The reasons for this expansion include:

- The addition of the pointers that link the objects together
- Translation of character data into Unicode, which can double the size
- The inclusion of field names that might have been implicit in the bit stream
- The presence of control data that is associated with the broker's operation

Manipulating a large message tree can require a lot of storage. If you design a message flow that handles large messages that are made up of repeating structures, you can code ESQL statements that help to reduce the storage load on the broker. These statements support both random and sequential access to the message, but assume that you do not need access to the whole message at one time.

These ESQL statements cause the broker to perform limited parsing of the message, and to keep in storage at one time, only that part of the message tree that reflects a single record. If your processing requires you to retain information from record to record (for example, to calculate a total price from a repeating structure of items in an order), you can either declare, initialize, and maintain ESQL variables, or you can save values in another part of the message tree; for example, in LocalEnvironment.

This technique reduces the memory that is used by the broker to that needed to hold the full input and output bit streams, plus that needed for the message trees of just one record. This technique also provides memory savings when even a small number of repeats is encountered in the message. The broker uses partial parsing and the ability to parse specified parts of the message tree, to and from the corresponding part of the bit stream.

To use these techniques in your Compute node, apply these general techniques:

- Copy the body of the input message as a bit stream to a special folder in the output message. The copy creates a modifiable copy of the input message that is not parsed and that therefore uses a minimum amount of memory.
- Avoid any inspection of the input message, which avoids the need to parse the message.
- Use a loop and a reference variable to step through the message one record at a time. For each record:
  - Use normal transforms to build a corresponding output subtree in a second special folder.
  - Use the ASBITSTREAM function to generate a bit stream for the output subtree. The generated bit stream is stored in a BitStream element that is placed in the position in the output subtree that corresponds to its required position in the final bit stream.
  - Use the DELETE statement to delete both the current input and output record message trees when you have completed their manipulation.
  - When you have completed the processing of all records, detach the special folders so that they do not appear in the output bit stream.

You can vary these techniques to suit the processing that is required for your messages.

The following ESQL code provides an example of one implementation, and is a modification of the ESQL example in "Transforming a complex message" on page 388. It uses a single SET statement with nested SELECT functions to transform a message that contains nested, repeating structures.

```
-- Copy the MQMD header
  SET OutputRoot.MQMD = InputRoot.MQMD;

-- Create a special folder in the output message to hold the input tree
-- Note : SourceMessageTree is the root element of an XML parser
  CREATE LASTCHILD OF OutputRoot.XMLNS.Data DOMAIN 'XMLNS' NAME 'SourceMessageTree';

-- Copy the input message to a special folder in the output message
-- Note : This is a root to root copy which will therefore not build trees
  SET OutputRoot.XMLNS.Data.SourceMessageTree = InputRoot.XMLNS;

-- Create a special folder in the output message to hold the output tree
  CREATE FIELD OutputRoot.XMLNS.Data.TargetMessageTree;

-- Prepare to loop through the purchased items
  DECLARE sourceCursor REFERENCE TO OutputRoot.XMLNS.Data.SourceMessageTree.Invoice;
  DECLARE targetCursor REFERENCE TO OutputRoot.XMLNS.Data.TargetMessageTree;
  DECLARE resultCursor REFERENCE TO OutputRoot.XMLNS.Data;
  DECLARE grandTotal   FLOAT     0.0e0;

-- Create a block so that it's easy to abandon processing
  ProcessInvoice: BEGIN
  -- If there are no Invoices in the input message, there is nothing to do
    IF NOT LASTMOVE(sourceCursor) THEN
      LEAVE ProcessInvoice;
    END IF;

  -- Loop through the invoices in the source tree
  InvoiceLoop : LOOP
    -- Inspect the current invoice and create a matching Statement
    SET targetCursor.Statement = THE (SELECT 'Monthly' AS (XML.Attribute)Type,
                                            'Full' AS (0x03000000)Style[1],
                                            I.Customer.FirstName AS Customer.Name,
                                            I.Customer.LastName AS Customer.Surname,
                                            (SELECT
                                              FIELDVALUE(II.Title) AS Title,
                                              CAST(II.UnitPrice AS FLOAT) * 1.6 AS Cost,
                                              II.Quantity AS Qty
                                            FROM I.Purchases.Item[] AS II
                                            WHERE II.UnitPrice> 0.0) AS Purchases.Article[],
                                            (SELECT
                                              SUM( CAST(II.UnitPrice AS FLOAT) *
                                                   CAST(II.Quantity  AS FLOAT) *
                                                   1.6                         )
                                            FROM I.Purchases.Item[] AS II) AS Amount,
                                            'Dollars' AS Amount.(XML.Attribute)Currency
                                          FROM sourceCursor AS I
                                          WHERE I.Customer.LastName <> 'White');

    -- Turn the current Statement into a bit stream
    DECLARE StatementBitStream BLOB
    ASBITSTREAM(targetCursor.Statement OPTIONS FolderBitStream);
    -- If the SELECT produced a result
    -- (that is, it was not filtered out by the WHERE clause),
    -- process the Statement
    IF StatementBitStream IS NOT NULL THEN
    -- create a field to hold the bit stream in the result tree
       CREATE LASTCHILD OF resultCursor
          Type  XML.BitStream
          NAME  'StatementBitStream'
          VALUE StatementBitStream;
```

```
   -- Add the current Statement's Amount to the grand total
   -- Note that the cast is necessary because of the behavior
   -- of the XML syntax element
      SET grandTotal = grandTotal
       + CAST(targetCursor.Statement.Amount AS FLOAT);
   END IF;

   -- Delete the real Statement tree leaving only the bit stream version
   DELETE FIELD targetCursor.Statement;

   -- Step onto the next Invoice,
   -- removing the previous invoice and any
   -- text elements that might have been
   -- interspersed with the Invoices

   REPEAT
     MOVE sourceCursor NEXTSIBLING;
     DELETE PREVIOUSSIBLING OF sourceCursor;
     UNTIL (FIELDNAME(sourceCursor) = 'Invoice')
      OR (LASTMOVE(sourceCursor) = FALSE)
   END REPEAT;

   -- If there are no more invoices to process, abandon the loop
   IF NOT LASTMOVE(sourceCursor) THEN
    LEAVE InvoiceLoop;
   END IF;

  END LOOP InvoiceLoop;
END ProcessInvoice;

-- Remove the temporary source and target folders
DELETE FIELD OutputRoot.XMLNS.Data.SourceMessageTree;
DELETE FIELD OutputRoot.XMLNS.Data.TargetMessageTree;

-- Finally add the grand total
SET resultCursor.GrandTotal = grandTotal;
```

This ESQL code produces the following output message:

```
<Data>
 <Statement Type="Monthly" Style="Full">
  <Customer>
   <Name>Andrew</Name>
   <Surname>Smith</Surname>
   <Title>Mr</Title>
  </Customer>
  <Purchases>
   <Article>
    <Title>The XML Companion</Title>
    <Cost>4.472E+1</Cost>
    <Qty>2</Qty>
   </Article>
   <Article>
    <Title>A Complete Guide to DB2 Universal Database</Title>
    <Cost>6.872E+1</Cost>
    <Qty>1</Qty>
   </Article>
   <Article>
    <Title>JAVA 2 Developers Handbook</Title>
    <Cost>9.5984E+1</Cost>
    <Qty>1</Qty>
   </Article>
  </Purchases>
  <Amount Currency="Dollars">2.54144E+2</Amount>
  </Statement>
  <GrandTotal>2.54144E+2</GrandTotal>
 </Data>
```

Manipulating messages in the XMLNSC domain:

This topic provides information about on how to write ESQL for processing messages that belong to the XMLNSC domain, and that are parsed by the XMLNSC parser. Refer to "XMLNSC parser" on page 104 for background information.

The following topics provide detailed information about the structure of the message tree that the XMLNSC parser builds, and the field types that it uses.

- "XMLNSC: The XML declaration"
- "XMLNSC: The inline DTD" on page 409
- "XMLNSC: The message body" on page 410
- "XMLNSC: XML Schema support" on page 418

Information for users who are migrating from XML, XMLNS, or MRM XML is available in "Migrating to XMLNSC" on page 420.

Further information about processing XML messages can be found in "Working with XML messages" on page 398.

XMLNSC: The XML declaration:

The XML declaration is represented in the message tree by a syntax element with field type XMLNSC.XMLDeclaration.

If an XML declaration is created by the XMLNSC parser, its name is 'XmlDeclaration'. However, when a message tree is being produced, the name is not important: the XMLNSC parser recognizes this syntax element by its field type only. The following example shows a typical declaration:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>........</s1>
```

The XML Declaration has three optional attributes; Version, Standalone, and Encoding. The XMLNSC parser does not define special field types for these attributes. Instead, they are identified by their name, and by their position as a child of the XML Declaration element.

ESQL example code to create an XML declaration

To construct the XML declaration that is shown in the previous example, code the following ESQL statements:

```
CREATE FIRSTCHILD OF OutputRoot.XMLNSC TYPE XMLNSC.XmlDeclaration;
SET OutputRoot.XMLNSC.(XMLNSC.XmlDeclaration)*.(XMLNSC.Attribute)Version = '1.0';
SET OutputRoot.XMLNSC.(XMLNSC.XmlDeclaration)*.(XMLNSC.Attribute)Encoding = 'UTF-8';
SET OutputRoot.XMLNSC.(XMLNSC.XmlDeclaration)*.(XMLNSC.Attribute)StandAlone = 'yes';
```

The first line is optional; if it is omitted, the XMLNSC.XMLDeclaration element is automatically created when it is referenced by the second line.

Java example code to create an XML declaration

To construct the XML declaration that is shown in the previous example, write the following Java code:

```
//Create the XML domain root node
MBElement xmlRoot =
 root.createElementAsLastChild(MbXMLNSC.PARSER_NAME);
//Create the XML declaration parent node
MbElement xmlDecl =
 xmlRoot.createElementAsFirstChild(MbXMLNSC.XML_DECLARATION);

xmlDecl.setName("XmlDeclaration");

MbElement version =
 xmlDecl.CreateElementAsFirstChild(MbXMLNSC.ATTRIBUTE, "Version", "1.0");
MbElement encoding =
 xmlDecl.CreateElementAsFirstChild(MbXMLNSC.ATTRIBUTE, "Encoding", "utf-8");
MbElement standalone =
 xmlDecl.CreateElementAsFirstChild(MbXMLNSC.ATTRIBUTE, "Standalone", "yes");
```

Note: In both the ESQL example and the Java example, 'Version', 'StandAlone', and 'Encoding' can all be written in lowercase.

XMLNSC: The inline DTD:

When parsing an XML document that has an inline DTD, the XMLNSC parser does not put the DTD information into the message tree. However, using ESQL you can add XML entity definitions to the message tree, and these are used when the message tree is produced by the XMLNSC parser.

ESQL example code for entity definition and entity reference

This example assumes that InputRoot.XMLNSC has been created from the following XML message:

```
<BookInfo dtn="BookInfo" edn="author" edv="A.N.Other"/>
```

The following output message is generated:

```
<!DOCTYPE BookInfo [<!ENTITY author "A.N.Other">]>
<BookInfo><entref>&author;</entref></BookInfo>
```

The ESQL to create the output message is:

```
DECLARE cursor REFERENCE TO InputRoot.XMLNSC.BookInfo;
DECLARE docTypeName CHARACTER cursor.dtn;
DECLARE authorRef CHARACTER 'author';
-- Create <!DOCTYPE BOOKInfo ...
SET OutputRoot.XMLNSC.(XMLNSC.DocumentType)* NAME = docTypeName;
-- Create  <!ENTITY author "A.N.Other" > ...
SET OutputRoot.XMLNSC.(XMLNSC.DocumentType){docTypeName}.(XMLNSC.EntityDefinition) {authorRef} =
cursor.edv;
-- Create the entity reference
SET OutputRoot.XMLNSC.(XMLNSC.Folder){docTypeName}.(XMLNSC.EntityReference)entref = authorRef;
```

Java example code to create an XML declaration

To construct the XML declaration that is shown above, the following Java is required:

```
//Create the XML domain root node
MBElement xmlRoot =
 root.createElementAsLastChild(MbXMLNSC.PARSER_NAME);
//Create the XML declaration parent node
MbElement xmlDecl =
 xmlRoot.createElementAsFirstChild(MbXMLNSC.XML_DECLARATION);

xmlDecl.setName("XmlDeclaration");
```

```
MbElement version =
 xmlDecl.CreateElementAsFirstChild(MbXMLNSC.ATTRIBUTE, "Version", "1.0");
MbElement encoding =
 xmlDecl.CreateElementAsFirstChild(MbXMLNSC.ATTRIBUTE, "Encoding", "utf-8");
MbElement standalone =
 xmlDecl.CreateElementAsFirstChild(MbXMLNSC.ATTRIBUTE, "Standalone", "Yes");
```

XMLNSC: The message body:

The XMLNSC parser builds a message tree from the body of an XML document.

The following topics describe how the XMLNSC parser builds a message tree from the body of an XML document:

- "XMLNSC: Using field types" on page 107
- "XMLNSC: Attributes and elements" on page 412
- "XMLNSC: Namespace declarations" on page 413
- "XMLNSC: Element values and mixed content" on page 415
- "XMLNSC: Comments and Processing Instructions" on page 417

XMLNSC: Using field types:

The XMLNSC parser sets the field type on every syntax element that it creates.

The field type indicates the type of XML construct that the element represents. The XMLNSC parser uses the field type when writing a message tree. The field type can be set by using ESQL or Java to control the output XML. The field types that are used by the XMLNSC parser must be referenced by using constants with names that are prefixed by 'XMLNSC.'

Tip: Field type constants that have the prefix 'XML.' are for use with the XMLNS and XML parsers only, and are not valid with the XMLNSC or MRM parsers.

Field types for creating syntax elements

Use the following field type constants to create syntax elements in the message tree. The XMLNSC parser uses these values when creating a message tree from an input message.

XML construct	XMLNSC Field Type constant	Value
Simple Element	XMLNSC.Field XMLNSC.CDataField	0x03000000 0x03000001
Attribute	XMLNSC.SingleAttribute XMLNSC.Attribute	0x03000101 0x03000100
Mixed content	XMLNSC.Value XMLNSC.CDataValue	0x02000000 0x02000001
Namespace Declaration	XMLNSC.SingleNamespaceDecl XMLNSC.NamespaceDecl	0x03000102 0x03000103
Complex element	XMLNSC.Folder	0x01000000
Inline DTD	XMLNSC.DocumentType	0x01000300
XML declaration	XMLNSC.XmlDeclaration	0x01000400
Entity reference	XMLNSC.EntityReference	0x02000100
Entity definition	XMLNSC.SingleEntityDefinition XMLNSC.EntityDefinition	0x03000301 0x03000300

XML construct	XMLNSC Field Type constant	Value
Comment	XMLNSC.Comment	0x03000400
Processing Instruction	XMLNSC.ProcessingInstruction	0x03000401

Field types for path expressions (generic field types)

Use the following field type constants when querying the message tree by using a path expression; for example:

```
SET str = FIELDVALUE(InputRoot.e1.(XMLNSC.Attribute)attr1)
```

It is good practice to specify field types when querying a message tree built by the XMLNSC parser. This makes your ESQL code more specific and more readable, and it avoids incorrect results in some cases. However, care is required when choosing which field type constant to use. When you use the XMLNSC parser, use a generic field type constant when querying the message tree. This allows your path expression to tolerate variations in the input XML.

The generic field type constants are listed in the following table:

XML construct	XMLNSC Field Type constant	Purpose
Tag	XMLNSC.Element	Matches any tag, whether it contains child tags (XMLNSC.Folder) or a value (XMLNSC.Field)
Element	XMLNSC.Field	Matches a tag which contains normal text, CData, or a mixture of both. Does not match tags which contain child tags.
Attribute	XMLNSC.Attribute	Matches single-quoted and double-quoted attributes
Mixed content	XMLNSC.Value	Matches normal text, CData, or a mixture of both
XML Declaration	XMLNSC.NamespaceDecl	Matches single- and double-quoted declarations

If you write

```
InputRoot.e1.(XMLNSC.DoubleAttribute)attrName
```

your path expression does not match a single-quoted attribute. If you use the generic field type constant *XMLNSC.Attribute*, your message flow works with either single-quoted or double-quoted attributes.

Note that you should always use the field type constants and not their numeric values.

Field types for controlling output format

The following field types are provided for XML Schema and base64 support. Do not use these field type constants in path expressions; use them in conjunction with XMLNSC.Attribute and XMLNSC.Field to indicate the required output format for DATE and BLOB values. See “XMLNSC: XML Schema support” on page 418 for further information.

XMLNSC Field Type constant	Purpose	Value
XMLNSC.gYear	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gYear format.	0x00000010
XMLNSC.gYearMonth	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gYearMonth format.	0x00000040
XMLNSC.gMonth	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gMonth format.	0x00000020
XMLNSC.gMonthDay	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gMonthDay format.	0x00000050
XMLNSC.gDay	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gDay format.	0x00000030
XMLNSC.base64Binary	The value must be a BLOB. The value is produced with base64 encoding.	0x00000060
XMLNSC.List	The element must be XMLNSC.Attribute or XMLNSC.Field. If the field type includes this value, the values of all child elements in the message tree are produced as a space-separated list.	0x00000070

Field types for direct output

Use the following field types to produce pre-constructed segments of an XML document. Character escaping is not done; therefore, take extra care not to construct a badly-formed output document. Use these constants only after carefully exploring alternative solutions.

XMLNSC Field Type constant	Purpose	Value
XMLNSC.Bitstream	The value of this syntax element must be a BLOB. The value is written directly to the output bit stream. For more information about its usage, see “Working with large XML messages” on page 404.	0x03000200
XMLNSC.AsisElementContent	The value of this syntax element must be CHARACTER. The value is written directly to the output bit stream. No character substitutions are performed. Use this element with care.	0x03000600

XMLNSC: Attributes and elements:

The XMLNSC parser uses field types to represent attributes and elements.

Use the following field type constants when creating your own syntax elements in the message tree.

Table 9. Specific field type constants

XML Construct	XMLNSC Field Type constant	Value
Complex element	XMLNSC.Folder	0x01000000
Simple element	XMLNSC.Field XMLNSC.CDataField	0x02000000 0x02000001
Attribute	XMLNSC.SingleAttribute XMLNSC.Attribute	0x03000100 0x03000101

When accessing elements and attributes in the message tree, use generic field type constants which match all of the alternative values. Because there is only one type of Folder element, it is safe to use XMLNSC.Folder when querying the message tree.

Table 10. Generic field type constants

XML Construct	XMLNSC Field Type constant	Purpose
Element	XMLNSC.Field	Matches elements that contain normal text, CData, or a mixture of both
Attribute	XMLNSC.Attribute	Matches both single-quoted and double-quoted attributes

ESQL code examples

The following examples use this XML message:

```
<root id="12345">
    <id>ABCDE</id>
</root>
```

Note that the message contains an attribute and an element with the same name.

Example 1 : Query the value of an XML element

```
SET value = FIELDVALUE(InputRoot.XMLNSC.root.(XMLNSC.Field)id)
```

The result is that value is set to 'ABCDE'.

Example 2 : Query the value of an XML attribute

```
SET value = FIELDVALUE(InputRoot.XMLNSC.root.(XMLNSC.Attribute)id)
```

The result is that value is set to '12345'.

Example 3 : Create the example message by using ESQL

```
CREATE LASTCHILD OF OutputRoot.XMLNSC Type 'XMLNSC.Folder' Name 'root';
-- Note : XMLNSC.Attribute could be used here as well.
SET OuputRoot.XMLNSC.root.(XMLNSC.Attribute)id = '12345';
SET OuputRoot.XMLNSC.root.(XMLNSC.Field)id = 'ABCDE';
```

The first line is optional because the element 'root' is created automatically by the following line if it does not already exist.

XMLNSC: Namespace declarations:

The XMLNSC parser provides full support for namespaces.

The XMLNSC parser sets the correct namespace on every syntax element that it creates while parsing a message, and stores namespace declarations in the message tree. The parser uses the namespace declarations to select the appropriate prefixes when outputting the message tree.

The XMLNSC parser uses the following field types to represent namespace declarations. Use the field type constants that are listed in this table when you create namespace declarations in the message tree.

Table 11. Specific field type constants

XML construct	XMLNSC field type constant	Value
Namespace declaration	XMLNSC.SingleNamespaceDecl XMLNSC.DoubleNamespaceDecl	0x03000102 0x03000103

When accessing elements and attributes in the message tree, do not use the constants that are listed in the previous table; instead, use the generic field type constant that matches both of the values in the table above.

Table 12. Generic field type constants

XML construct	XMLNSC field type constant	Purpose
Namespace declaration	XMLNSC.NamespaceDecl	Matches namespace declarations in both single and double quotation marks

ESQL code examples

Example 1 : Declaring a non-empty prefix

```
DECLARE space1 NAMESPACE 'namespace1';
SET OutputRoot.XMLNSC.space1:root.(XMLNSC.NamespaceDecl)xmlns:ns1 = space1;
SET OutputRoot.XMLNSC.space1:root.space1:example = 'ABCDE';
```

This creates the following XML message:

```
<ns1:root xmlns:ns1="namespace1">
    <ns1:example>ABCDE</ns1:example>
</ns1:root>
```

Note that the NAMESPACE constant `space1` is just a local variable in the ESQL; it does not affect the namespace prefix `ns1` that is defined by the NameSpaceDecl element and appears in the output message.

However, as shown here, `space1` can be used to initialize the NameSpaceDecl for `ns1`. This avoids the need to duplicate the namespace URI ('namespace1' in this example), which in practice is typically a much longer string.

Example 2 : Declaring an empty prefix

```
DECLARE space1 NAMESPACE 'namespace1';
SET OutputRoot.XMLNSC.space1:root.(XMLNSC.NamespaceDecl)xmlns = space1;
SET OutputRoot.XMLNSC.space1:root.space1:example = 'ABCDE';
```

This creates the following XML message:

```
<root xmlns="namespace1">
    <example>ABCDE</example>
</root>
```

Note that the syntax elements `root` and `example` must have a non-empty namespace. The default namespace declaration means that any child element without a prefix is in the namespace `namespace1`.

Example 3 : Example of incorrect usage

```
DECLARE space1 NAMESPACE 'namespace1';
SET OutputRoot.XMLNSC.root.(XMLNSC.NamespaceDecl)xmlns = space1;
SET OutputRoot.XMLNSC.root.example = 'ABCDE';
```

This example causes the XMLNSC parser to issue the message BIP5014 when it attempts to create the message tree. The elements `root` and `example` are both within the scope of the default namespace declaration; therefore, in ESQL, these elements must be qualified by a namespace prefix bound to that namespace.

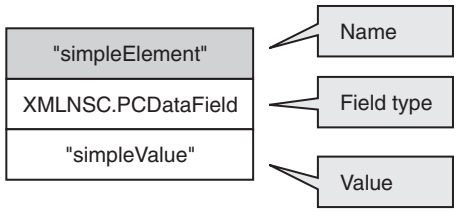
XMLNSC: Element values and mixed content:

The XMLNSC parser is a compact parser; therefore, an element with single content is parsed as a single syntax element. When an element has both child elements and some text, the text is called *mixed content*.

Element with simple content

The following XML fragment with a single content is parsed as a single syntax element:

```
<simpleElement>simpleValue</simpleElement>
```



The value of this element can be queried with this ESQL:

```
SET val = FIELDVALUE(InputRoot.XMLNSC.(XMLNSC.Field)simpleElement);
```

To generate an element with simple content in the output:

```
SET OutputRoot.XMLNSC.(PCDataField)simpleElement VALUE = 'simpleValue';
```

Note that XMLNSC.Field is used when querying the message tree, but XMLNSC.PCDataField is specified when constructing the output message. XMLNSC.PCDataField can be used to query the message tree; however, that would not work if the input message used a CData section, as shown in the following example:

```
<simpleElement><![CDATA[simpleValue]]></simpleElement>
```

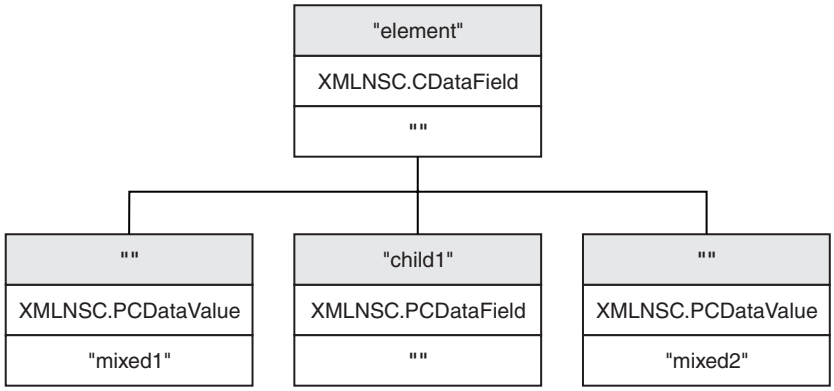
Element with mixed content

If an element has child elements, it is typically a 'folder', and does not have a value. When an element has both child elements and some text, the text is called 'mixed content'.

```
<element>mixed1<child/>mixed2</element>
```

By default, mixed content is discarded because it is typically just formatting white space and has no business meaning. Mixed content can be preserved if you select the Retain mixed content check box on the Parser Options page of the node properties.

If mixed content is being preserved, the XMLNSC parser creates a Value child element for each distinct item of mixed content.



The mixed content can be queried with this ESQL:

```
SET mixed1 = FIELDVALUE(InputRoot.XMLNSC.(element).*[1];
```

The ESQL to construct the above XML fragment is:

```
CREATE ref REFERENCE TO OutputRoot.XMLNSC.element;
CREATE FIRSTCHILD OF ref TYPE XMLNSC.PCDataValue VALUE 'mixed1';
CREATE LASTCHILD OF ref NAME 'child1' TYPE XMLNSC.PCDataField  VALUE '';
CREATE LASTSTCHILD OF ref TYPE XMLNSC.PCDataValue VALUE 'mixed2';
```

The following ESQL enables the *Retain mixed content* option:

```
DECLARE X BLOB;
-- assume that X contains an XML document
CREATE LASTCHILD OF OutputRoot
    PARSE(X OPTIONS XMLNSC.MixedContentRetainAll);
```

Element containing a CData section

A CData section is an XML notation that allows XML markup characters to be included in the content of an element.

The following two XML fragments are identical in their meaning:

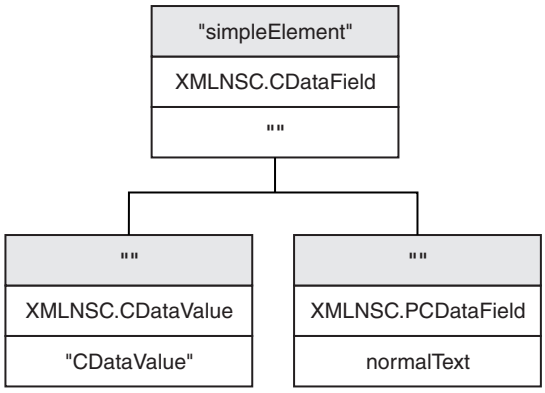
```
<simpleElement>simpleValue</simpleElement>
```

```
<simpleElement><![CDATA[simpleValue]]></simpleElement>
```

If the CData section is the only text content, the XMLNSC parser remembers that the input document contained a CData section by setting the field type to XMLNSC.CDataField instead of XMLNSC.PCDataField.

If the CData section is not the only text content, it is created as a child value element, with other child value elements representing the remaining text content. The following is an example of this:

```
<simpleElement><![CDATA[CDataValue]]>normalText</simpleElement>
```



See "XMLNSC: Working with CData" on page 402 for more information about the correct use of CData in XML documents.

XMLNSC: Comments and Processing Instructions:

The XMLNSC parser discards comments and processing instructions because both comments and processing instructions are auxiliary information with no business meaning.

Comments

Comments can be preserved if you select the *Retain comments* check box on the Parser Options page of the node properties.

If *Retain comments* is selected, each comment in the input document is represented by a single syntax element with field type XMLNSC.Comment. The *Retain comments* option can also be accessed by using the following ESQL:

```
DECLARE X BLOB;
-- assume that X contains an XML document
CREATE LASTCHILD OF OutputRoot.XMLNSC
    PARSE(X DOMAIN XMLNSC
          NAME preserveComments
          OPTIONS XMLNSC.CommentsRetainAll);

-- do it again, this time discarding comments
CREATE LASTCHILD OF OutputRoot.XMLNSC
    PARSE(X DOMAIN XMLNSC
          NAME discardComments
          OPTIONS XMLNSC.CommentsRetainNone);
```

Processing Instructions

Processing instructions can be preserved if you select the *Retain processing instructions* check box on the Parser Options page of the node properties.

If *Retain processing instructions* is selected, each processing instruction the input document is represented by a single syntax element with field type XMLNSC.ProcessingInstruction. The *Retain processing instructions* option can also be accessed by using the following ESQL:

```
DECLARE X BLOB;
-- assume that X contains an XML document
CREATE LASTCHILD OF OutputRoot.XMLNSC
    PARSE(X DOMAIN XMLNSC
          NAME preserveProcessingInstructions
          OPTIONS XMLNSC.ProcessingInstructionsRetainAll);

-- do it again, this time discarding processing instructions
CREATE LASTCHILD OF OutputRoot.XMLNSC
    PARSE(X DOMAIN XMLNSC
          NAME discardProcessingInstructions
          OPTIONS XMLNSC.ProcessingInstructionsRetainNone);
```

XMLNSC: XML Schema support:

Use the XMLNSC parser to parse and validate by using an XML Schema.

For information about how to configure the XMLNSC parser to use an XML Schema, see "XMLNSC parser" on page 104.

The following topics describe how the XMLNSC parser uses the field type to hold information about XML Schema simple types. This behavior enables the parser to write dates and binary elements in the same form in which they were parsed, and according to the XML Schema specification. It also allows the message flow developer to write dates, lists, and binary data in the correct XML Schema format.

- "XMLNSC: base64 support"
- "XMLNSC: XML Schema date formatting"
- "XMLNSC: XML List type support" on page 419

XMLNSC: base64 support:

The XMLNSC parser can produce binary data in base64-encoded format.

If Validation is set to Content and Value, and Build tree using schema types is enabled, the XMLSNC parser automatically decodes base64 data and creates a BLOB value in the message tree. When producing a message tree, the XMLNSC parser will 'base64-encode' a BLOB if the field type includes the constant *XMLNSC.base64Binary*.

ESQL code example to output base64 data

```
DECLARE Base64Data BLOB '0102030405060708090A0B0C0D0E0F';
-- Add in the base64Binary field type
DECLARE base64FieldType INTEGER XMLNSC.Field + XMLNSC.base64Binary;
CREATE LASTCHILD OF OutputRoot DOMAIN 'XMLNSC' NAME 'XMLNSC';
CREATE LASTCHILD OF OutputRoot.XMLNSC TYPE base64FieldType NAME 'myBinaryData' VALUE Base64Data;
```

Result : <myBinaryData>AQIDBAUGBwgJCgsMDQ4P</myBinaryData>

Note that this example does not depend on validation. The XMLNSC parser can produce base64 binary data even if Validation is set to None.

XMLNSC: XML Schema date formatting:

The XMLNSC parser can parse and write all of the XML Schema simple types.

The rarely used types gYear, gYearMonth, gMonth, gMonthDay, and gDay do not map directly to a message broker data type. For these simple types, the XMLNSC

parser adds one of the following constant values to the field type. This behavior allows the parser to produce the data for output in the same format as it was received.

Field types for controlling date format

The following field types are provided for XML Schema date format support. Do not use these field type constants in path expressions. Use them in conjunction with constants XMLNSC.Attribute and XMLNSC.Field to indicate the required output format for DATE values.

XMLNSC Field Type constant	Purpose	Value
XMLNSC.gYear	Value must be a DATE. If the field type includes this value, the DATE value is written by using the XML Schema gYear format.	0x00000010
XMLNSC.gYearMonth	Value must be a DATE. If the field type includes this value, the DATE value is written by using the XML Schema gYearMonth format.	0x00000040
XMLNSC.gMonth	Value must be a DATE. If the field type includes this value, the DATE value is written by using the XML Schema gMonth format.	0x00000020
XMLNSC.gMonthDay	Value must be a DATE. If the field type includes this value, the DATE value is written by using the XML Schema gMonthDay format.	0x00000050
XMLNSC.gDay	Value must be a DATE. If the field type includes this value, the DATE value is written by using the XML Schema gDay format.	0x00000030

ESQL code example

```
DECLARE gYear DATE '2007-01-01';
-- Add in the gYear field type
DECLARE gYearFieldType INTEGER XMLNSC.Field + XMLNSC.gYear;
CREATE LASTCHILD OF OutputRoot DOMAIN 'XMLNSC' NAME 'XMLNSC';
CREATE LASTCHILD OF OutputRoot.XMLNSC TYPE gYearFieldType NAME 'gYear' VALUE gYear;
```

Result : <gYear>2007</gYear>

XMLNSC: XML List type support:

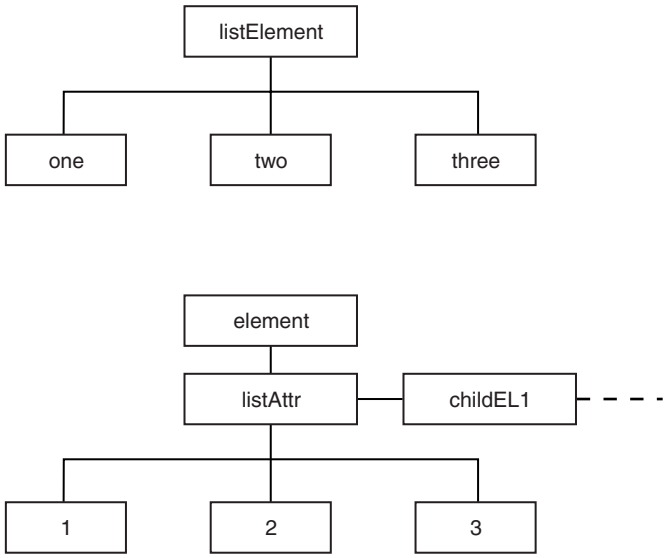
The XMLNSC parser can automatically parse a space-separated list of values into individual syntax elements in the message tree if you select certain options.

An element or an attribute can have multiple values separated by spaces, as shown in the following examples:

```
<listElement>one two three</listElement>
```

```
<element listAttribute="1 2 3"><childEL1/></element>
```

If your XML schema specifies a list type for an element or an attribute, and Validation is set to Content and Value, and Build tree using schema types is enabled, the XMLNSC parser automatically parses the space-separated list of values into individual syntax elements in the message tree. The resulting message tree looks like this:
and for an attribute with a list value it looks like this:



ESQL code examples

Access the individual values in a list

```
SET val = InputRoot.XMLNSC.listElement.*[1];
```

Result : val = 'one'

```
SET val = InputRoot.XMLNSC.element.(XMLNSC.Attribute)listAttr.*[3];
```

Result : val='3'

Create a list element in the message tree

```
CREATE LASTCHILD OF OutputRoot.XMLNSC
   Name 'listElement'
   Type XMLNSC.List;
DECLARE listEl REFERENCE TO OutputRoot.XMLNSC.listElement;
DECLARE listValType INTEGER XMLNSC.PCDataValue;
CREATE LASTCHILD OF listEl TYPE listValType VALUE 'one';
CREATE LASTCHILD OF listEl TYPE listValType VALUE 'two';
CREATE LASTCHILD OF listEl TYPE listValType VALUE 'three';
```

Migrating to XMLNSC:

The XMLNSC parser now offers the best combination of features and performance for most applications.

Reasons to migrate

If your message flow uses the XMLNS or XML domain, you might want to migrate a message flow to XMLNSC to take advantage of the XML schema validation. If

your message flow uses the MRM domain, you might want to migrate to XMLNSC to obtain standards-compliant validation, and a large reduction in CPU usage.

Migrating from the XMLNS or XML domain

The XMLNSC parser differs from the XMLNS parser in the following ways:

- The XMLNSC parser builds a compact message tree.
- It uses different field type constants.
- It discards inline DTDs

In most cases, the compact message tree has no affect on ESQL paths or XPath expressions. Typically, a simple message tree query produces the same results in XMLNSC as in the XMLNS or XML domain. Merely changing the correlation name from XMLNS to XMLNSC is usually sufficient, but care must be taken with complex XPath expressions that navigate to the value of an element, then to its parent in a single query. These might produce different results in the XMLNSC domain.

The field type constants that are used by the XMLNSC parser are completely different from those used by XMLNS or XML. Every occurrence of XML.Attribute, XML.XmlDecl, for example, must be changed to use the equivalent XMLNSC field type constant.

The discarding of inline DTDs only affects message flows that process the DTD.

Migrating from MRM XML

The XMLNSC parser differs from the MRM XML parser in the following ways:

- The XMLNSC parser uses field types to identify the XML constructs in the message tree. The MRM parser distinguishes attributes from elements by matching the message tree against the message definition.
- When writing a message tree, the XMLNSC parser selects namespace prefixes by detecting and using xmlns attributes in the message tree. The MRM XML parser uses a table in the message set properties.
- The MRM parser does not include the root tag of the XML document in the message tree.

Migrating a message flow from MRM to XMLNSC typically requires extensive changes to your message flow. However, the migration usually delivers a large reduction in CPU usage, and allows much more accurate control of the output XML.

Manipulating messages in the XMLNS domain:

This topic provides information about how to write ESQL for processing messages that belong to the XMLNS domain, and that are parsed by the XMLNS parser. Most of the information in this topic, and the topics mentioned within it, also applies to the XML domain, unless it refers to features that are not supported in the XML domain.

Refer to “XMLNS parser” on page 115 for background information.

The following topics provided detailed information about the structure of the message tree that the XMLNS parser builds, and the field types that it uses.

- "XMLNS: The XML declaration"
- "XMLNS: The DTD" on page 423
- "XMLNS: The XML message body" on page 424

More information about processing XML messages can be found in "Working with XML messages" on page 398.

XMLNS: The XML declaration:

The beginning of an XML message can contain an XML declaration.

The following is an example of a declaration:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>........</s1>
```

The XML Declaration is represented by the following types of syntax element:

- "XML.XMLDecl"
- "XML.version"
- "XML.encoding"
- "XML.standalone" on page 423

"XMLNS: XML declaration example" on page 423 includes another example of an XML declaration and the tree structure that it forms.

XML.XMLDecl:

The XML Declaration is represented in the message tree by a syntax element with field type 'XML.XMLDecl'.

If the XML declaration is created by the XMLNS parser its name is 'XMLDecl'. However, when a message tree is being written, the name is not important; only the field type is used by the parser.

The following shows an example of a declaration:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>........</s1>
```

XML.version:

The XML version attribute in the XML declaration is represented in the message tree by a syntax element with field type 'XML.version'.

The value of the XML version attribute is the value of the version attribute. It is always a child of an XML.XmlDecl syntaxelement. In the following example, the version element contains the string value 1.0:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>........</s1>
```

XML.encoding:

The encoding attribute is represented by a syntax element with field type 'XML.encoding', and is always a child of an XML.XmlDecl syntax element.

In the following example, the encoding attribute has a value of UTF-8.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>........</s1>
```

You cannot specify WebSphere MQ encodings in this element.

In your ESQL, (XML,"Encoding") must include quotation marks, because Encoding is a reserved word.

XML.standalone:

The XML standalone element defines the existence of an externally-defined DTD. In the message tree it is represented by a syntax element with field type XML.standalone.

The value of the XML standalone element is the value of the standalone attribute in the XML declaration. It is always a child of an XML.XmlDecl syntax element. The only valid values for the standalone element are yes and no. The following is an example of this:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>........</s1>
```

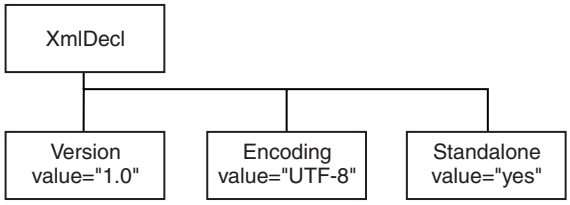
A value of no indicates that this XML document is not standalone, and depends on an externally-defined DTD. A value of yes indicates that the XML document is self-contained. However, because the current release of WebSphere Message Broker does not resolve externally-defined DTDs, the setting of standalone is irrelevant, and is ignored.

XMLNS: XML declaration example:

The following example shows an XML declaration in an XML document:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

The following figure shows the tree structure that the XMLNS parser creates for this declaration:



XMLNS: The DTD:

The document type declaration (DTD) of an XML message is represented by a syntax element with field type XML.DocTypeDecl, and its children. These comprise the DOCTYPE construct.

Only internal (inline) DTD subsets are represented in the syntax element tree. An inline DTD is a DTD that is declared within the XML document itself. It can be a complete DTD definition, or it can extend the definition in an external DTD.

External DTD subsets (identified by the SystemID or PublicId elements described later in this section) can be referenced in the message, but those referenced are not resolved by the broker.

The following field type constants can be used to reference the various parts of a DTD in the message tree:

- "XML DocTypeDecl" on page 1574
- "XML NotationDecl" on page 1575
- "XML entities" on page 1575
- "XML ElementDef" on page 1577
- "XML AttributeList" on page 1577
- "XML AttributeDef" on page 1577
- "XML DocTypePI" on page 1578
- "XML WhiteSpace and DocTypeWhiteSpace" on page 1579
- "XML DocTypeComment" on page 1578

"XML DTD example" on page 1579 shows an example of an XML DTD.

See "XML document type declaration" on page 1574 for more information about handling an inline DTD.

XMLNS: The XML message body:

Every XML message must have a body. The body consists of a hierarchy of XML elements and other XML constructs that represent the message data.

The XMLNS parser assigns a field type to every syntax element that it creates. The value of the field type indicates the XML construct that it represents. In the following topics, each field type is discussed, with a series of example XML fragments.

The following common element types are discussed:

- "XML.element" on page 427
- "XML.Attribute" on page 427
- "XML.content" on page 427

"XML message body example" on page 431 provides an example of an XML message body and the tree structure that is created from it using the syntax elements types that are listed above.

More complex XML messages might use some of the following syntax element types:

- "XML.CDataSection" on page 427
- "XML.EntityReferenceStart and XML.EntityReferenceEnd" on page 428
- "XML.comment" on page 429
- "XML.ProcessingInstruction" on page 429
- "XML.AsisElementContent" on page 429
- "XML.BitStream" on page 430

Accessing attributes and elements:

The XMLNS parser sets the field type on every message tree element that it creates.

The field type indicates the type of XML construct that the element represents. The field types used by the XMLNS parser can be referenced using constants with names prefixed with 'XML.' Field type constants with prefix 'XML.' are for use with the XMLNS and XML parsers only; they do not work with the XMLNSC or MRM parsers.

	XMLNS Field Type constant
Tag	XML.Element
Attribute	XML.Attribute XML.Attr

By using the field type in this way, the XMLNS parser can distinguish between an element and an attribute that have the same name.

Example XML

```
<parent id="12345">
    <id>ABCDE</id>
</parent>
```

Example ESQL

```
SET value = FIELDVALUE(InputRoot.XMLNS.parent.(XML.Element)id)
Result : value is 'ABCDE'

SET value = FIELDVALUE(InputRoot.XMLNS.parent.(XML.Attr)id)
Result : value is '12345'
```

Example using SELECT to access multiple attributes

In the example Invoice message, the element Title within each Item element has three attributes: Category, Form, and Edition. For example, the first Title element contains:

```
<Title Category="Computer" Form="Paperback" Edition="2">The XML Companion</Title>
```

The element `InputRoot.XML.Invoice.Purchases.Item[1].Title` has four children in the logical tree: Category, Form, Edition, and the element value, which is "The XML Companion".

If you want to access the attributes for this element, you can code the following ESQL. This extract of code retrieves the attributes from the input message and creates them as elements in the output message. It does not process the value of the element itself in this example.

```
-- Set the cursor to the first XML.Attribute of the Title.
-- Note the * after (XML.Attribute) meaning any name, because the name might not be known
DECLARE cursor REFERENCE TO InputRoot.XMLNS.Invoice.Purchases.Item[1].Title.(XML.Attribute)*;
WHILE LASTMOVE(cursor) DO
-- Create a field with the same name as the XML.Attribute
-- and set its value to the value of the XML.Attribute
  SET OutputRoot.XML.Data.Attributes.{FIELDNAME(cursor)} = FIELDVALUE(cursor);
-- Move to the next sibling of the same TYPE to avoid the Title value
-- which is not an XML.Attribute
  MOVE cursor NEXTSIBLING REPEAT TYPE;
END WHILE;
```

When this ESQL is processed by the Compute node, the following output message is generated:

```
<Data>
  <Attributes>
    <Category>Computer</Category>
    <Form>Paperback</Form>
    <Edition>2</Edition>
  </Attributes>
</Data>
```

You can also use a SELECT statement:

```
SET OutputRoot.XMLNS.Data.Attributes[] =
  (SELECT FIELDVALUE(I.Title) AS title,
          FIELDVALUE(I.Title.(XML.Attribute)Category) AS category,
          FIELDVALUE(I.Title.(XML.Attribute)Form) AS form,
          FIELDVALUE(I.Title.(XML.Attribute)Edition) AS edition
     FROM InputRoot.XML.Invoice.Purchases.Item[] AS I);
```

This statement generates the following output message:

```
<Data>
  <Attributes>
    <title>The XML Companion</title>
    <category>Computer</category>
    <form>Paperback</form>
    <edition>2</edition>
  </Attributes>
  <Attributes>
    <title>A Complete Guide to DB2 Universal Database</title>
    <category>Computer</category>
    <form>Paperback</form>
    <edition>2</edition>
  </Attributes>
  <Attributes>
    <title>JAVA 2 Developers Handbook</title>
    <category>Computer</category>
    <form>Hardcover</form>
    <edition>0</edition>
  </Attributes>
</Data>
```

You can qualify the SELECT with a WHERE statement to narrow down the results to obtain the same output message as the one that is generated by the WHILE statement. This second example shows that you can create the same results with less, and less complex, ESQL.

```
SET OutputRoot.XMLNS.Data.Attributes[] =
  (SELECT FIELDVALUE(I.Title.(XML.Attribute)Category) AS category,
          FIELDVALUE(I.Title.(XML.Attribute)Form) AS form,
          FIELDVALUE(I.Title.(XML.Attribute)Edition) AS edition
     FROM InputRoot.XML.Invoice.Purchases.Item[] AS I)
     WHERE I.Title = 'The XML Companion');
```

This statement generates the following output message:

```
<Data>
  <Attributes>
    <Category>Computer</Category>
    <Form>Paperback</Form>
    <Edition>2</Edition>
  </Attributes>
  </Data>
```

XML.element: This syntax element represents an XML element (a tag). The name of the syntax element corresponds to the name of the XML element in the message. This element can have many children in the message tree, including attributes, elements, and content.

XML.tag is supported as an alternative to XML.element for compatibility with earlier versions of WebSphere Message Broker. Use XML.element in any new message flows that you create.

XML.Attribute: **Parsing**

The XMLNS parser uses this field type for syntax elements that represent an XML attribute. The name and value of the syntax element correspond to the name and value of the XML attribute that is represented. Attribute elements have no children, and must always be children of an element.

Writing

When the XMLNS parser generates a bit stream from a message tree, occurrences of ampersand (&), less than (<), greater than (>), double quotation mark ("), and apostrophe ('), within the attribute value, are replaced by the predefined XML entities &, <, >, ", and '.

The XML.attr field type constant is also supported for compatibility with earlier versions of WebSphere Message Broker.

XML.content: The XMLNS parser uses this syntax element to represent character data (including an XML attribute). The name and value of the syntax element correspond to the name and value of the XML attribute that is represented. Attribute elements have no children, and must always be children of an element.

Writing

When the XMLNS parser generates a bit stream from a message tree, occurrences of ampersand (&), less than (<), greater than (>), double quotation mark ("), and apostrophe ('), within the attribute value, are replaced by the predefined XML entities &, <, >, ", and '.

XML.CDataSection:

CData sections in the XML message are represented by a syntax element with field type XML.CdataSection.

The content of the CDataSection element is the value of the CDataSection element without the `<![CDATA[` that marks its beginning, and the `]]>` that marks its end.

For example, the following CData section:

```
<![CDATA[<greeting>Hello, world!</greeting>]]>
```

is represented by a CDataSection element with a string value of:

```
"<greeting>Hello, world!</greeting>"
```

Unlike Content, occurrences of <,>, &, ", and ' are not translated to their XML character entities (<, >, and &) when the CDataSection is produced.

When to use XML.CDataSection

A CData section is often used to embed one XML message within another. By using a CData section, you ensure that the XML reserved characters (<, >, and &) are not replaced with XML character entities.

XML.AsisElementContent also allows the production of unmodified character data, but XML.CDataSection is typically a better choice because it protects the outer message from errors in the embedded message.

Parsing the contents of a CDataSection

A common requirement is to parse the contents of a CData section to create a message tree, which you can achieve by using the ESQL statement CREATE with the PARSE clause; see "XMLNSC: Working with XML messages and bit streams" on page 403.

XML.NamespaceDecl:

A namespace declaration is represented by a syntax element with field type XML.NamespaceDecl.

Namespace declarations take one of two forms in the message tree:

- **Declaration using a namespace prefix**

  ```
  <ns1:e1 xmlns:ns1="namespace1"/>
  ```

  In the message tree, the syntax element for this namespace declaration is shown in the following table:

Namespace	`http://www.w3.org/2000/xmlns/`
Name	`ns1`
Value	`namespace1`

- **Declaration of a default namespace**

  A default namespace declaration is an xmlns attribute that defines an empty prefix

  ```
  <e1 xmlns="namespace1"/>
  ```

  In the message tree, the syntax element for this namespace declaration is shown in the following table:

Namespace	""
Name	`xmlns`
Value	`namespace1`

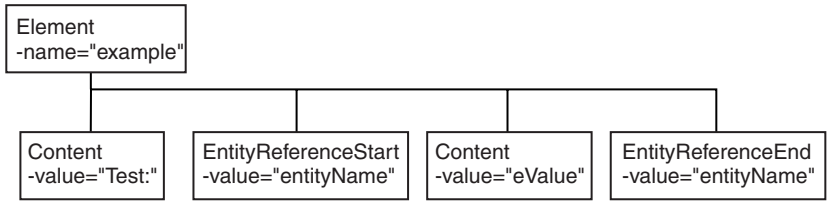
  Note that, in both cases, element 'e1' is in namespace 'namespace1'.

XML.EntityReferenceStart and XML.EntityReferenceEnd:

When an entity reference is encountered in the XML message, both the expanded form and the original entity name are stored in the syntax element tree. The name of the entity is stored as the value of the EntityReferenceStart and EntityReferenceEnd syntax elements, and any syntax elements between contain the entity expansion.

The following examples show the XML entity references in an XML document, and in tree structure form.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE example [ <!ENTITY entityName "eValue"> ]>
<example>Test: &entityName;</example>
```



The XML declaration and the document type declaration are not shown here. Refer to "XMLNS: The XML declaration" on page 422 and "XMLNS: The DTD" on page 423 for details of those sections of the syntax element tree.

XML.comment:

An XML.comment that is encountered outside the document type declaration is represented by a syntax element with field type XML.comment. The value of the element is the comment text from the XML message.

If the value of the element contains the character sequence `-->`, the sequence is replaced with the text `-->`. This ensures that the contents of the comment cannot prematurely terminate the comment. Occurrences of the following characters are not translated to their escape sequences:

```
< > & " '
```

The following is an example of the XML comment in an XML document:

```
<example><!-- This is a comment --></example>
```

XML.ProcessingInstruction:

A processing instruction that is encountered outside the document type declaration is represented by a syntax element with field type XML.ProcessingInstruction.

This is a name-value element; the name of the syntax element is the processing instruction target name, and the value of the syntax element is the character data of the processing instruction. The value of the syntax element must not be empty. The name cannot be `XML` in either uppercase or lowercase.

If the value of the element contains the character sequence `?>`, the sequence is replaced with the text `?>`. This ensures that the content of the processing instruction cannot prematurely terminate the processing instruction. Occurrences of the following characters are not translated to their escape sequences:

```
< > & " '
```

The following shows an example of the XML processing instruction in an XML document:

```
<example><?target This is a PI.?></example>
```

XML.AsisElementContent:

Use the special field type XML.AsisElementContent to precisely control generated XML.

XML.AsisElementContent is a special field type. Use the field type in a message flow to precisely control the XML that is generated in an output message, without the safeguards of the Element, Attribute, and Content syntax elements. The XMLNS parser never creates elements with this field type.

Try to avoid using AsisElementContent; there is typically a safer alternative approach. If you do use AsisElementContent, it is your responsibility to ensure that the output message is well-formed XML.

You might choose to use AsisElementContent if, for example, you want to suppress the usual behavior in which occurrences of ampersand (&), less than (<), greater than (>), double quotation mark ("), and apostrophe (') are replaced by the predefined XML entities &, <, >, ", and '.

The following example illustrates the use of AsisElementContent. The statement:

```
Set OutputRoot.XMLNS.(XML.Element)Message.(XML.Content) = '<rawMarkup>';
```

generates the following XML in an output message:

```
<Message>&lt;rawMarkup&gt;</Message>
```

However, the statement:

```
Set OutputRoot.XMLNS.(XML.Element)Message.(XML.AsisElementContent) = '<rawMarkup>';
```

generates the following XML in an output message:

```
<Message><rawMarkup></Message>
```

This shows that the value of an AsisElementContent syntax element is not modified before it is written to the output message.

XML.BitStream:

BitStream is a specialized element designed to aid the processing of very large messages.

XML.Bitstream is a special field type. When writing an XML message, the value of the BitStream element is written directly into the message, and the name is not important. The BitStream element might be the only element in the message tree.

The value of the element must be of type BLOB; any other data type generates an error when writing the element. Ensure that the content of the element is appropriate for use in the output message; pay special attention to the CCSID and the encoding of the XML text in the BLOB.

Use of the BitStream element is similar to the use of the AsisElementContent element, except that the AsisElementContent type converts its value into a string, whereas the BitStream element uses its BLOB value directly. This is a specialized element designed to aid the processing of very large messages.

The following ESQL excerpts demonstrate some typical use of this element. First, declare the element:

```
DECLARE StatementBitStream BLOB
```

Initialize the contents of StatementBitStream from an appropriate source, such as an input message. If the source field is not of type BLOB, use the CAST statement to convert the contents to BLOB. Then create the new field in the output message; for example:

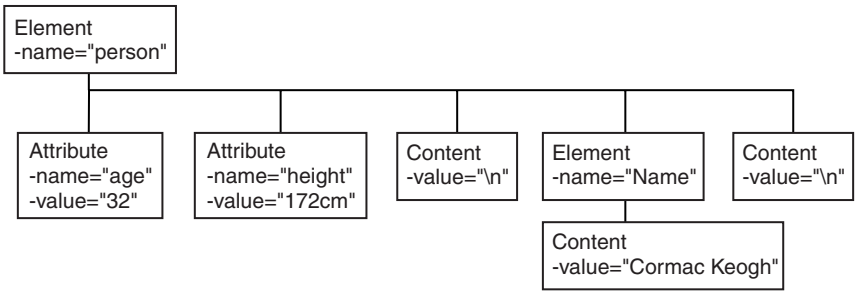
```
CREATE LASTCHILD OF resultCursor
 Type XML.BitStream
 NAME 'StatementBitStream'
 VALUE StatementBitstream;
```

XML message body example:

The XMLNS parser creates a message tree that represents an XML document.

The following example shows the message tree that the XMLNS parser creates for the following snippet from a simple XML document:

```
<Person age="32" height="172cm">
<Name>Cormac Keogh</Name>
</Person>
```



Manipulating messages in the XML domain:

The XML parser is very similar to the XMLNS parser, but it has no support for namespaces or opaque parsing. For information about how to work with the XML parser, refer to "Manipulating messages in the XMLNS domain" on page 421.

Manipulating messages in the MRM domain

How to use messages that have been modeled in the MRM domain, and that are parsed by the MRM parser.

The following topics show you how to deal with messages that have been modeled in the MRM domain, and that are parsed by the MRM parser. The physical formats associated with the message models do not affect this information unless specifically stated. Use this information in conjunction with the information about manipulating message body content; see "Manipulating message body content" on page 327.

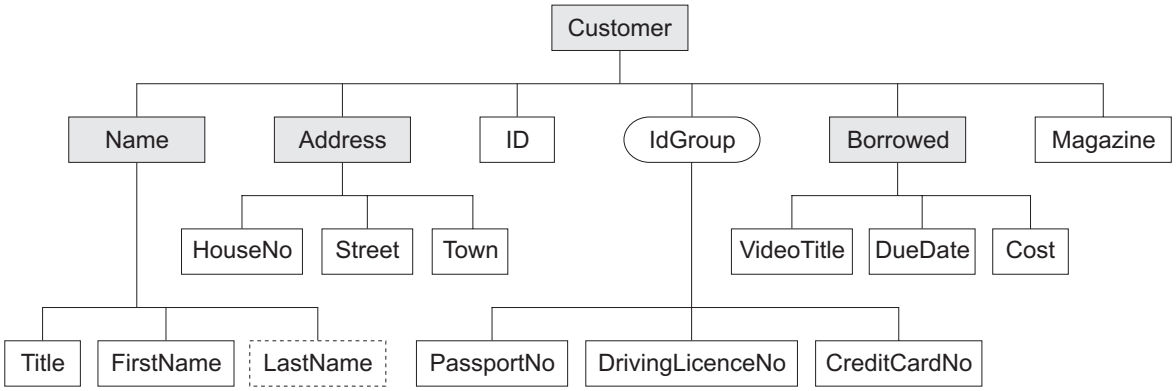
- "Querying null values in a message in the MRM domain" on page 441
- "Setting null values in a message in the MRM domain" on page 441
- "Working with MRM messages and bit streams" on page 442
- "Handling large MRM messages" on page 445

The structure of the "Customer" message is shown in the following sample:

- Video Rental

The message is used in the samples in the topics listed previously to show ESQL that manipulates the objects that can be defined in a message model.

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



The message includes a variety of structures that demonstrate how you can classify metadata to the MRM. Within an MRM message set, you can define the following objects: messages, types, groups, elements, and attributes. Folder icons that represent each of these types of objects are displayed for each message definition file in the Broker Application Development perspective.

Each message definition file can contribute to a namespace; in this sample, each namespace is completely defined by a single message definition file. You can combine several message definition files to form a complete message dictionary, which you can then deploy to a broker.

The video sample has three message definition files:

Customer.mxsd
Resides in the `no target` namespace

Address.mxsd
Resides in the namespace http://www.ibm.com/AddressDetails

Borrowed.mxsd
Resides in the namespace http://www.ibm.com/BorrowedDetails

Look at the video rental message structure sample for detailed information about the objects that are defined in this message model:

- Video Rental

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

Accessing elements in a message in the MRM domain:

You can use ESQL to manipulate the logical tree that represents a message in the message flow. This topic describes how to access data for elements in a message in the MRM domain.

You can populate an element with data with the SET statement:

```
SET OutputRoot.MRM.Name = UPPER(InputRoot.MRM.Name);
```

The field reference on the left hand side of the expression refers to the element called Name within the MRM message domain. This statement takes the input value for the Name field, converts it to uppercase, and assigns the result to the same element in the output message.

The Name element is defined in the noTarget namespace. No namespace prefix is specified in front of the Name part of the field reference in the example above. If you have defined an MRM element in a namespace other than the noTarget namespace, you must also specify a namespace prefix in the statement. For example:

```
DECLARE brw NAMESPACE 'http://www.ibm.com/Borrowed';

SET OutputRoot.MRM.brw:Borrowed.VideoTitle = 'MRM Greatest Hits';
```

For more information about using namespaces with messages in the MRM domain, see "Accessing the content of a message in the MRM domain with namespace support enabled" on page 440.

Accessing multiple occurrences of an element in a message in the MRM domain:

You can use specific ESQL code to set the value of one occurrence of an element that has multiple occurrences in a message. You can also use arrow notation to indicate the direction of search when searching for multiple occurrences of an element.

You can access MRM domain elements following the general guidance given in "Accessing known multiple occurrences of an element" on page 332 and "Accessing unknown multiple occurrences of an element" on page 333. Further information specific to MRM domain messages is provided in this topic.

Consider the following statements:

```
DECLARE brw NAMESPACE 'http://www.ibm.com/Borrowed';

SET OutputRoot.MRM.brw:Borrowed[1].VideoTitle = 'MRM Greatest Hits Volume 1';
SET OutputRoot.MRM.brw:Borrowed[2].VideoTitle = 'MRM Greatest Hits Volume 2';
```

The above SET statements operate on two occurrences of the element Borrowed. Each statement sets the value of the child VideoTitle. The array index indicates which occurrence of the repeating element you are interested in.

When you define child elements of a complex type (which has its Composition property set to Sequence) in a message set, you can add the same element to the complex type more than once. These instances do not have to be contiguous, but you must use the same method (array notation) to refer to them in ESQL.

For example, if you create a complex type with a Composition of Sequence that contains the following elements:

- `StringElement1`

- IntegerElement1
- StringElement1

use the following ESQL to set the value of StringElement1:

```
SET OutputRoot.MRM.StringElement1[1] =
             'This is the first occurrence of StringElement1';
SET OutputRoot.MRM.StringElement1[2] =
             'This is the second occurrence of StringElement1';
```

You can also use the arrow notation (the greater than (>) and less than (<) symbols) to indicate the direction of search and the index to be specified:

```
SET OutputRoot.MRM.StringElement1[>] =
              'This is the first occurrence of StringElement1';
SET OutputRoot.MRM.StringElement1[<2] =
              'This is the last but one occurrence of
 StringElement1';
SET OutputRoot.MRM.StringElement1[<1] =
              'This is the last occurrence of StringElement1';
```

Refer to "Accessing known multiple occurrences of an element" on page 332 and "Accessing unknown multiple occurrences of an element" on page 333 for additional detail.

Accessing attributes in a message in the MRM domain:

When an MRM message is parsed into a logical tree, attributes and the data that they contain are created as name-value pairs in the same way that MRM elements are. The ESQL that you code to interrogate and update the data held in attributes refers to the attributes in a similar manner.

Consider the Video Rental sample MRM message. The attribute LastName is defined as a child of the Name element in the Customer message. Here is an example input XML message:

```
<Customer xmlns:addr="http://www.ibm.com/AddressDetails"

xmlns:brw="http://www.ibm.com/BorrowedDetails">
   <Name LastName="Bloggs">
      <Title>Mr</Title>
      <FirstName>Fred</FirstName>
   </Name>
   <addr:Address>
      <HouseNo>13</HouseNo>
      <Street>Oak Street</Street>
      <Town>Southampton</Town>
   </addr:Address>
            <ID>P</ID>
   <PassportNo>J123456TT</PassportNo>
   <brw:Borrowed>
      <VideoTitle>Fast Cars</VideoTitle>
      <DueDate>2003-05-23T01:00:00</DueDate>
      <Cost>3.50</Cost>
   </brw:Borrowed>
   <brw:Borrowed>
      <VideoTitle>Cut To The Chase</VideoTitle>
      <DueDate>2003-05-23T01:00:00</DueDate>
      <Cost>3.00</Cost>
   </brw:Borrowed>
   <Magazine>0</Magazine>
</Customer>
```

When the input message is parsed, values are stored in the logical tree as shown in the following section of user trace:

```
(0x0100001B):MRM = (
  (0x01000013):Name = (
    (0x0300000B):LastName = 'Bloggs'
    (0x0300000B):Title = 'Mr'
    (0x0300000B):FirstName = 'Fred'
  )
  (0x01000013)http://www.ibm.com/AddressDetails:Address = (
    (0x0300000B):HouseNo = 13
    (0x0300000B):Street = 'Oak Street'
    (0x0300000B):Town = 'Southampton'
  )
  (0x0300000B):ID = 'P'
  (0x0300000B):PassportNo = 'J123456TT'
  (0x01000013)http://www.ibm.com/BorrowedDetails:Borrowed = (
    (0x0300000B):VideoTitle = 'Fast Cars'
    (0x0300000B):DueDate = TIMESTAMP '2003-05-23 00:00:00'
    (0x0300000B):Cost = 3.50
  )
  (0x01000013)http://www.ibm.com/BorrowedDetails:Borrowed = (
    (0x0300000B):VideoTitle = 'Cut To The Chase '
    (0x0300000B):DueDate = TIMESTAMP '2003-05-23 00:00:00'
    (0x0300000B):Cost = 3.00
  )
  (0x0300000B):Magazine = FALSE
```

The following ESQL changes the value of the LastName attribute in the output message:

```
SET OutputRoot.MRM.Name.LastName = 'Smith';
```

Be aware of the ordering of attributes when you code ESQL. When attributes are parsed, the logical tree inserts the corresponding name-value before the MRM element's child elements. In the previous example, the child elements Title and FirstName appear in the logical message tree after the attribute LastName. In the

Broker Application Development perspective, the Outline view displays attributes after the elements. When you code ESQL to construct output messages, you must define name-value pairs for attributes before any child elements.

The following sample shows the structure of the Customer message:

- Video Rental

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

Accessing elements within groups in a message in the MRM domain:

When an input message is parsed, structures that you have defined as groups in your message set are not represented in the logical tree, but its children are. If you want to refer to or update values for elements that are children of a groups, do not include the group in the ESQL statement. Groups do not have tags that appear in instance messages, and do not appear in user trace of the logical message tree.

Consider the following Video message:

```
<Customer xmlns:addr="http://www.ibm.com/AddressDetails"
xmlns:brw="http://www.ibm.com/BorrowedDetails">
   <Name LastName="Bloggs">
      <Title>Mr</Title>
      <FirstName>Fred</FirstName>
   </Name>
   <addr:Address>
      <HouseNo>13</HouseNo>
      <Street>Oak Street</Street>
      <Town>Southampton</Town>
   </addr:Address>
             <ID>P</ID>
   <PassportNo>J123456TT</PassportNo>
   <brw:Borrowed>
      <VideoTitle>Fast Cars</VideoTitle>
      <DueDate>2003-05-23T01:00:00</DueDate>
      <Cost>3.50</Cost>
   </brw:Borrowed>
   <brw:Borrowed>
      <VideoTitle>Cut To The Chase</VideoTitle>
      <DueDate>2003-05-23T01:00:00</DueDate>
      <Cost>3.00</Cost>
   </brw:Borrowed>
   <Magazine>0</Magazine>
</Customer>
```

When the input message is parsed, values are stored in the logical tree as shown in the following section of user trace:

```
(0x0100001B):MRM = (
  (0x01000013):Name = (
    (0x0300000B):LastName = 'Bloggs'
    (0x0300000B):Title = 'Mr'
    (0x0300000B):FirstName = 'Fred'
  )
  (0x01000013)http://www.ibm.com/AddressDetails:Address = (
    (0x0300000B):HouseNo = 13
    (0x0300000B):Street = 'Oak Street'
    (0x0300000B):Town = 'Southampton'
  )
  (0x0300000B):ID = 'P'
  (0x0300000B):PassportNo = 'J123456TT'
  (0x01000013)http://www.ibm.com/BorrowedDetails:Borrowed = (
    (0x0300000B):VideoTitle = 'Fast Cars'
    (0x0300000B):DueDate = TIMESTAMP '2003-05-23 00:00:00'
    (0x0300000B):Cost = 3.50
  )
  (0x01000013)http://www.ibm.com/BorrowedDetails:Borrowed = (
    (0x0300000B):VideoTitle = 'Cut To The Chase '
    (0x0300000B):DueDate = TIMESTAMP '2003-05-23 00:00:00'
    (0x0300000B):Cost = 3.00
  )
  (0x0300000B):Magazine = FALSE
```

Immediately following the element named ID, the MRM message definition uses a group which has a *Composition* of `Choice`. The group is defined with three children: PassportNo, DrivingLicenceNo, and CreditCardNo. The choice composition dictates that instance documents must use only one of these three possible alternatives. The example shown above uses the PassportNo element.

When you refer to this element in ESQL statements, you do not specify the group to which the element belongs. For example:

```
SET OutputRoot.MRM.PassportNo = 'J999999TT';
```

If you define messages within message sets that include XML and TDS physical formats, you can determine from the message data which option of a choice has been taken, because the tags in the message represent one of the choice's options. However, if your messages have CWF physical format, or are non-tagged TDS messages, it is not clear from the message data, and the application programs processing the message must determine which option of the choice has been selected. This is known as *unresolved choice handling*. For further information, see the description of the value of `Choice` in Complex type logical properties.

Accessing mixed content in a message in the MRM domain:

When you define a complex type in a message model, you can optionally specify its content to be mixed. This setting, in support of mixed content in XML Schema, allows you to manipulate data that is included between elements in the message.

Consider the following example:

```
<MRM>
  <Mess1>
    abc
    <Elem1>def</Elem1>
    ghi
    <Elem2>jkl</Elem2>
    mno
    <Elem3>pqr</Elem3>
  </Mess1>
</MRM>
```

The strings abc, ghi, and mno do not represent the value of a particular element (unlike def, for example, which is the value of element Elem1). The presence of these strings means that you must model Mess1 with mixed content. You can model this XML message in the MRM using the following objects:

Message
The message *Name* property is set to `Mess1` to match the XML tag.

The *Type* property is set to `tMess1`.

Type The complex type *Name* property is set to `tMess1`.

The *Composition* property is set to `OrderedSet`.

The complex type has mixed content.

The complex type contains the following objects:

Element
The *Name* property is set to `Elem1` to match the XML tag.

The *Type* property is set to simple type `xsd:string`.

Element
The *Name* property is set to `Elem2` to match the XML tag.

The *Type* property is set to simple type `xsd:string`.

Element
The *Name* property is set to `Elem3` to match the XML tag.

The *Type* property is set to simple type `xsd:string`.

If you code the following ESQL:

```
SET OutputRoot.MRM.*[1] = InputBody.Elem3;
SET OutputRoot.MRM.Elem1 = InputBody.*[5];
SET OutputRoot.MRM.*[3] = InputBody.Elem2;
SET OutputRoot.MRM.Elem2 = InputBody.*[3];
SET OutputRoot.MRM.*[5] = InputBody.Elem1;
SET OutputRoot.MRM.Elem3 = InputBody*[1];
```

the mixed content is successfully mapped to the following output message:

```
<MRM>
  <Mess1>
    pqr
    <Elem1>mno</Elem1>
    jkl
    <Elem2>ghi</Elem2>
    def
    <Elem3>abc</Elem3>
  </Mess1>
</MRM>
```

Accessing embedded messages in the MRM domain:

If you have defined a multipart message, you have at least one message embedded within another. Within the overall complex type that represents the outer messages, you can model the inner message in two ways.

You can model the inner message in the following ways:

- An element (named E_outer1 in the following example) with its *Type* property set to a complex type that has been defined with its *Composition* property set to `Message`
- A complex type with its *Composition* property set to `Message` (named t_Embedded in the following example)

The ESQL that you need to write to manipulate the inner message varies depending on which of the above models you have used. For example, assume that you have defined:

- An outer message M_outer that has its *Type* property set to t_Outer.
- An inner message M_inner1 that has its *Type* set to t_Inner1
- An inner message M_inner2 that has its *Type* set to t_Inner2
- Type t_Outer that has its first child element named E_outer1 and its second child defined as a complex type named t_Embedded
- Type t_Embedded that has its *Composition* property set to `Message`
- Type t_Inner1 that has its first child element named E_inner11
- Type t_Inner2 that has its first child element named E_inner21
- Type t_outer1 that has its *Composition* property set to `Message`
- Element E_outer1 that has its *Type* property set to t_outer1

If you want to set the value of E_inner11, code the following ESQL:

```
SET OutputRoot.MRM.E_outer1.M_inner1.E_inner11 = 'FRED';
```

If you want to set the value of E_inner21, code the following ESQL:

```
SET OutputRoot.MRM.M_inner2.E_inner21 = 'FRED';
```

If you copy message headers from the input message to the output message, and your input message type contains a path, only the outermost name in the path is copied to the output message type.

When you configure a message flow to handle embedded messages, you can specify the path of a message type in either an MQRFH2 header (if one is present in the input message) or in the input node *Message Type* property in place of a name (for example, for the message modeled above, the path could be specified as M_Outer/M_Inner1/M_Inner2 instead of just M_Outer).

If you have specified that the input message has a physical format of either CWF or XML, any message type prefix is concatenated in front of the message type from the MQRFH2 or input node, giving a final path to use (for more information refer to Multipart messages). The MRM uses the first item in the path as the outermost message type, then progressively works inwards when it finds a complex type with its *Composition* property set to `Message`.

If you have specified that the input message has a physical format of TDS, a different process that uses message keys is implemented. This is described in TDS format: Multipart messages.

For more information about path concatenations, see Message set properties.

Accessing the content of a message in the MRM domain with namespace support enabled:

Use namespaces where appropriate for messages that are parsed by the MRM parser.

When you want to access elements of a message and namespaces are enabled, you must include the namespace when you code the ESQL reference to the element. If you do not do so, the broker searches the `no target` namespace. If the element is not found in the `no target` namespace, the broker searches all other known namespaces in the message dictionary (that is, within the deployed message set). For performance and integrity reasons, specify namespaces wherever they apply.

The most efficient way to refer to elements when namespaces are enabled is to define a namespace constant, and use this in the appropriate ESQL statements. This technique makes your ESQL code much easier to read and maintain.

Define a constant using the DECLARE NAMESPACE statement:

```
DECLARE ns01 NAMESPACE 'http://www.ns01.com'
.
.
SET OutputRoot.MRM.Element1 = InputBody.ns01:Element1;
```

`ns01` is interpreted correctly as a namespace because of the way that it is declared.

You can also use a CHARACTER variable to declare a namespace:

```
DECLARE ns02 CHARACTER 'http://www.ns02.com'
.
.
SET OutputRoot.MRM.Element2 = InputBody.{ns02}:Element2;
```

If you use this method, you must surround the declared variable with braces to ensure that it is interpreted as a namespace.

If you are concerned that a CHARACTER variable might get changed, you can use a CONSTANT CHARACTER declaration:

```
DECLARE ns03 CONSTANT CHARACTER 'http://www.ns03.com'
.
.
SET OutputRoot.MRM.Element3 = InputBody.{ns03}:Element3;
```

You can declare a namespace, constant, and variable within a module or function. However, you can declare only a namespace or constant in schema scope (that is, outside a module scope).

The following sample provides further examples of the use of namespaces:

- Video Rental

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

Querying null values in a message in the MRM domain:

You can use an ESQL statement to compare an element to NULL.

If you want to compare an element to NULL, code the statement:

```
IF InputRoot.MRM.Elem2.Child1 IS NULL THEN
   DO:
    -- more ESQL --
END IF;
```

If nulls are permitted for this element, this statement tests whether the element exists in the input message, or whether it exists and contains the MRM-supplied null value. The behavior of this test depends on the physical format:

- For an XML element, if the XML tag or attribute is not in the bit stream, this test returns TRUE.
- For an XML element, if the XML tag or attribute is in the bit stream and contains the MRM null value, this test returns TRUE.
- For an XML element, if the XML tag or attribute is in the bit stream and does not contain the MRM null value, this test returns FALSE.
- For a delimited TDS element, if the element has no value between the previous delimiter and its delimiter, this test returns TRUE.
- For a delimited TDS element, if the element has a value between the previous delimiter and its delimiter that is the same as the MRM-defined null value for this element, this test returns TRUE.
- For a delimited TDS element, if the element has a value between the previous delimiter and its delimiter that is not the MRM-defined null value, this test returns FALSE.
- For a CWF or fixed length TDS element, if the element's value is the same as the MRM-defined null value for this element, this test returns TRUE.
- For a CWF or fixed length TDS element, if the element's value is not the same as the MRM-defined null value, this test returns FALSE.

If you want to determine if the field is missing, rather than present but with null value, you can use the ESQL CARDINALITY function.

Setting null values in a message in the MRM domain:

You can use implicit or explicit null processing to set the value of an element to NULL in an output message.

To set a value of an element in an output message, you normally code an ESQL statement similar to the following:

```
SET OutputRoot.MRM.Elem2.Child1 = 'xyz';
```

or its equivalent statement:

```
SET OutputRoot.MRM.Elem2.Child1 VALUE = 'xyz';
```

If you set the element to a non-null value, these two statements give identical results. However, if you want to set the value to null, these two statements do not give the same result:

1. If you set the element to NULL using the following statement, the element is deleted from the message tree:

```
SET OutputRoot.MRM.Elem2.Child1 = NULL;
```

   The content of the output bit stream depends on the physical format:

   - For an XML element, neither the XML tag or attribute nor its value are included in the output bit stream.
   - For a Delimited TDS element, neither the tag (if appropriate) nor its value are included in the output bit stream. The absence of the element is typically conveyed by two adjacent delimiters.
   - For a CWF or Fixed Length TDS element, the content of the output bit stream depends on whether you have set the *Default Value* property for the element. If you have set this property, the default value is included in the bit stream. If you have not set the property, an exception is raised.

   This is called implicit null processing.
2. If you set the value of this element to NULL as follows:

```
SET OutputRoot.MRM.Elem2.Child1 VALUE = NULL;
```

   the element is not deleted from the message tree. Instead, a special value of NULL is assigned to the element. The content of the output bit stream depends on the settings of the physical format null-handling properties.

   This is called explicit null processing.

Setting a complex element to NULL deletes that element and all its children.

Working with MRM messages and bit streams:

When you use the ASBITSTREAM function or the CREATE FIELD statement with a PARSE clause you must consider various restrictions.

The ASBITSTREAM function

If you code the ASBITSTREAM function with the parser mode option set to *RootBitStream*, to parse a message tree to a bit stream, the result is an MRM document in the format specified by the message format that is built from the children of the target element in the normal way.

The target element must be a predefined message defined within the message set, or can be a self-defined message if you are using an XML physical format. This algorithm is identical to that used to generate the normal output bit stream. A well-formed bit stream obtained in this way can be used to re-create the original tree by using a CREATE statement with a PARSE clause.

If you code the ASBITSTREAM function with the parser mode option set to *FolderBitStream*, to parse a message tree to a bit stream, the generated bit stream is an MRM element built from the target element and its children. Unlike *RootBitStream* mode the target element does not have to represent a message; it can represent a predefined element within a message or self-defined element within a message.

So that the MRM parser can correctly parse the message, the path from the message to the target element within the message must be specified in the *Message Type*. The format of the path is the same as that used by message paths except that the message type prefix is not used.

For example, suppose the following message structure is used:

```
Message
    elem1
         elem11
         elem12
```

To serialize the subtree representing element `elem12` and its children, specify the message path `'message/elem1/elem12'` in the *Message Type*.

If an element in the path is qualified by a namespace, specify the namespace URI between {} characters in the message path. For example if element `elem1` is qualified by namespace `'http://www.ibm.com/temp'`, specify the message path as `'message/{http://www.ibm.com/temp}elem1/elem12'`

This mode can be used to obtain a bit stream description of arbitrary sub-trees owned by an MRM parser. When in this mode, with a physical format of XML, the XML bit stream generated is not enclosed by the 'Root Tag Name' specified for the Message in the Message Set. No XML declaration is created, even if not suppressed in the message set properties.

Bit streams obtained in this way can be used to re-create the original tree by using a CREATE statement with a PARSE clause (by using a mode of *FolderBitStream*).

The CREATE statement with a PARSE clause

If you code a CREATE statement with a PARSE clause, with the parser mode option set to *RootBitStream*, to parse a bit stream to a message tree, the expected bit stream is a normal MRM document. A field in the tree is created for each field in the document. This algorithm is identical to that used when parsing a bit stream from an input node

If you code a CREATE statement with a PARSE clause, with the parser mode option set to *FolderBitStream*, to parse a bit stream to a message tree, the expected bit stream is a document in the format specified by the Message Format, which is either specified directly or inherited. Unlike *RootBitStream* mode the root of the document does not have to represent an MRM message; it can represent a predefined element within a message or self-defined element within a message.

So that the MRM parser can correctly parse the message the path from the message to the target element within the message must be specified in the *Message Type*. The format of the message path is the same as that used for the ASBITSTREAM function described above.

Example of using the ASBITSTREAM function and CREATE statement with a PARSE clause in FolderBitStream mode

The following ESQL uses the message definition described above. The ESQL serializes part of the input tree by using the ASBITSTREAM function, then uses the CREATE statement with a PARSE clause to re-create the subtree in the output tree. The Input message and corresponding Output message are shown below the ESQL.

```
CREATE COMPUTE MODULE DocSampleFlow_Compute
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  CALL CopyMessageHeaders();

  -- Set the options to be used by ASBITSTREAM and CREATE ... PARSE
  -- to be FolderBitStream and enable validation
  DECLARE parseOptions INTEGER BITOR(FolderBitStream, ValidateContent,
                       ValidateValue, ValidateLocalError);

  -- Serialise the elem12 element and its children from the input bitstream
  -- into a variable
       DECLARE subBitStream BLOB
         ASBITSTREAM(InputRoot.MRM.elem1.elem12
           OPTIONS parseOptions
           SET 'DocSample'
           TYPE 'message/elem1/elem12'
           FORMAT 'XML1');

        -- Set the value of the first element in the output tree
        SET OutputRoot.MRM.elem1.elem11 = 'val11';

        -- Parse the serialized sub-tree into the output tree
        IF subBitStream IS NOT NULL THEN
            CREATE LASTCHILD OF OutputRoot.MRM.elem1
                PARSE ( subBitStream
                        OPTIONS parseOptions
                        SET 'DocSample'
                        TYPE 'message/elem1/elem12'
                        FORMAT 'XML1');
        END IF;

  -- Convert the children of elem12 in the output tree to uppercase
  SET OutputRoot.MRM.elem1.elem12.elem121 =
   UCASE(OutputRoot.MRM.elem1.elem12.elem121);

  SET OutputRoot.MRM.elem1.elem12.elem122 =
   UCASE(OutputRoot.MRM.elem1.elem12.elem122);

        -- Set the value of the last element in the output tree
        SET OutputRoot.MRM.elem1.elem13 = 'val13';

  RETURN TRUE;
 END;

 CREATE PROCEDURE CopyMessageHeaders() BEGIN
  DECLARE I INTEGER 1;
  DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
  WHILE I < J DO
   SET OutputRoot.*[I] = InputRoot.*[I];
   SET I = I + 1;
  END WHILE;
 END;

END MODULE;
```

Input message :

```
<message>
    <elem1>
        <elem11>value11</elem11>
        <elem12>
            <elem121>value121</elem121>
            <elem122>value122</elem122>
```

```
        </elem12>
        <elem13>value13</elem13>
    </elem1>
</message>
```

Output message :

```
<message>
    <elem1>
        <elem11>val11</elem11>
        <elem12>
            <elem121>VALUE121</elem121>
            <elem122>VALUE122</elem122>
        </elem12>
        <elem13>val13</elem13>
    </elem1
</message
```

Handling large MRM messages:

When an input bit stream is parsed, and a logical tree created, the tree representation of an MRM message is typically larger, and in some cases much larger, than the corresponding bit stream.

The reasons for this large size include:

- The addition of the pointers that link the objects together
- Translation of character data into Unicode that can double the original size
- The inclusion of field names that can be contained implicitly within the bit stream
- The presence of control data that is associated with the broker's operation

Manipulation of a large message tree can, therefore, demand a great deal of storage. If you design a message flow that handles large messages made up of repeating structures, you can code specific ESQL statements that help to reduce the storage load on the broker. These statements support both random and sequential access to the message, but assume that you do not need access to the whole message at one time.

These ESQL statements cause the broker to perform limited parsing of the message, and to keep only that part of the message tree that reflects a single record in storage at a time. If your processing requires you to retain information from record to record (for example, to calculate a total price from a repeating structure of items in an order), you can either declare, initialize, and maintain ESQL variables, or you can save values in another part of the message tree, for example LocalEnvironment.

This technique reduces the memory used by the broker to that needed to hold the full input and output bit streams, plus that required for one record's trees. It provides memory savings when even a small number of repeats is encountered in the message. The broker makes use of partial parsing, and the ability to parse specified parts of the message tree, to and from the corresponding part of the bit stream.

To use these techniques in your Compute node apply these general techniques:

- Copy the body of the input message as a bit stream to a special folder in the output message. This creates a modifiable copy of the input message that is not parsed and which therefore uses a minimum amount of memory.

- Avoid any inspection of the input message; this avoids the need to parse the message.
- Use a loop and a reference variable to step through the message one record at a time. For each record:
  - Use normal transforms to build a corresponding output subtree in a second special folder.
  - Use the ASBITSTREAM function to generate a bit stream for the output subtree that is stored in a *BitStream* element, placed in the position in the tree, that corresponds to its required position in the final bit stream.
  - Use the DELETE statement to delete both the current input and the output record message trees when you complete their manipulation.
  - When you complete the processing of all records, detach the special folders so that they do not appear in the output bit stream.

You can vary these techniques to suit the processing that is required for your messages. The following ESQL provides an example of one implementation.

The ESQL is dependant on a message set called `LargeMessageExanple` that has been created to define messages for both the Invoice input format and the Statement output format. A message called `AllInvoices` has been created that contains a global element called `Invoice` that can repeat one or more times, and a message called `Data` that contains a global element called `Statement` that can repeat one or more times.

The definitions of the elements and attributes have been given the correct data types, therefore, the CAST statements used by the ESQL in the XML example are no longer required. An XML physical format with name `XML1` has been created in the message set which allows an XML message corresponding to these messages to be parsed by the MRM.

When the `Statement` tree is serialized using the ASBITSTREAM function the *Message Set*, *Message Type*, and *Message Format* are specified as parameters. The *Message Type* parameter contains the path from the message to the element being serialized which, in this case, is `Data/Statement` because the `Statement` element is a direct child of the `Data` message.

The input message to the flow is the same Invoice example message used in other parts of the documentation except that it is contained between the tags:

```
       <AllInvoices> ....  </AllInvoices>
```

```
CREATE COMPUTE MODULE LargeMessageExampleFlow_Compute
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  CALL CopyMessageHeaders();
    -- Create a special folder in the output message to hold the input tree
    -- Note : SourceMessageTree is the root element of an MRM parser
    CREATE LASTCHILD OF OutputRoot.MRM DOMAIN 'MRM' NAME 'SourceMessageTree';

    -- Copy the input message to a special folder in the output message
    -- Note : This is a root to root copy which will therefore not build trees
    SET OutputRoot.MRM.SourceMessageTree = InputRoot.MRM;

    -- Create a special folder in the output message to hold the output tree
    CREATE FIELD OutputRoot.MRM.TargetMessageTree;

    -- Prepare to loop through the purchased items
    DECLARE sourceCursor REFERENCE TO OutputRoot.MRM.SourceMessageTree.Invoice;
    DECLARE targetCursor REFERENCE TO OutputRoot.MRM.TargetMessageTree;
```

```
DECLARE resultCursor REFERENCE TO OutputRoot.MRM;
DECLARE grandTotal   FLOAT     0.0e0;

-- Create a block so that it's easy to abandon processing
ProcessInvoice: BEGIN
 -- If there are no Invoices in the input message, there is nothing to do
 IF NOT LASTMOVE(sourceCursor) THEN
    LEAVE ProcessInvoice;
 END IF;

 -- Loop through the invoices in the source tree
 InvoiceLoop : LOOP
    -- Inspect the current invoice and create a matching Statement
    SET targetCursor.Statement =
      THE (
         SELECT
           'Monthly'                        AS Type,
           'Full'                           AS Style,
           I.Customer.FirstName             AS Customer.Name,
           I.Customer.LastName              AS Customer.Surname,
           I.Customer.Title                 AS Customer.Title,
           (SELECT
              FIELDVALUE(II.Title)    AS Title,
              II.UnitPrice * 1.6    AS Cost,
              II.Quantity                AS Qty
           FROM I.Purchases.Item[] AS II
           WHERE II.UnitPrice> 0.0               ) AS Purchases.Article[],
           (SELECT
              SUM( II.UnitPrice *
                       II.Quantity  *
                     1.6                        )
            FROM I.Purchases.Item[] AS II                    ) AS Amount,
            'Dollars'                                  AS Amount.Currency
          FROM sourceCursor AS I
          WHERE I.Customer.LastName <> 'White'
       );

    -- Turn the current Statement into a bit stream
    -- The SET parameter is set to the name of the message set
     -- containing the MRM definition
    -- The TYPE parameter contains the path from the from the message
     -- to element being serialized
    -- The FORMAT parameter contains the name of the physical format
     -- name defined in the message
    DECLARE StatementBitStream BLOB
      ASBITSTREAM(targetCursor.Statement
       OPTIONS FolderBitStream
       SET 'LargeMessageExample'
       TYPE 'Data/Statement'
       FORMAT 'XML1');

    -- If the SELECT produced a result (that is, it was not filtered
     -- out by the WHERE clause), process the Statement
    IF StatementBitStream IS NOT NULL THEN
      -- create a field to hold the bit stream in the result tree
      -- The Type of the element is set to MRM.BitStream to indicate
       -- to the MRM Parser that this is a bitstream
      CREATE LASTCHILD OF resultCursor
         Type  MRM.BitStream
         NAME  'Statement'
         VALUE StatementBitStream;

      -- Add the current Statement's Amount to the grand total
      SET grandTotal = grandTotal + targetCursor.Statement.Amount;
    END IF;

    -- Delete the real Statement tree leaving only the bit stream version
```

```
        DELETE FIELD targetCursor.Statement;

        -- Step onto the next Invoice, removing the previous invoice and any
        -- text elements that might have been interspersed with the Invoices
        REPEAT
          MOVE sourceCursor NEXTSIBLING;
          DELETE PREVIOUSSIBLING OF sourceCursor;
        UNTIL (FIELDNAME(sourceCursor) = 'Invoice')
               OR (LASTMOVE(sourceCursor) = FALSE)
        END REPEAT;

        -- If there are no more invoices to process, abandon the loop
        IF NOT LASTMOVE(sourceCursor) THEN
          LEAVE InvoiceLoop;
        END IF;

     END LOOP InvoiceLoop;
    END ProcessInvoice;

    -- Remove the temporary source and target folders
    DELETE FIELD OutputRoot.MRM.SourceMessageTree;
    DELETE FIELD OutputRoot.MRM.TargetMessageTree;

    -- Finally add the grand total
    SET resultCursor.GrandTotal = grandTotal;

    -- Set the output MessageType property to be 'Data'
    SET OutputRoot.Properties.MessageType = 'Data';

  RETURN TRUE;
 END;

 CREATE PROCEDURE CopyMessageHeaders() BEGIN
  DECLARE I INTEGER 1;
  DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
  WHILE I < J DO
   SET OutputRoot.*[I] = InputRoot.*[I];
   SET I = I + 1;
  END WHILE;
 END;

END MODULE;
```

Manipulating messages in the JMS domains

This topic provides information specific to dealing with messages that belong to the JMSMap and JMSStream domains. These messages are parsed by the generic XML parser.

Messages that belong to the JMS domains are processed by the XML parser, so you can follow the guidance provided for XML messages in “Manipulating messages in the XML domain” on page 431, in conjunction with the information in “Manipulating message body content” on page 327.

The JMSMap and JMSStream domains support MapMessage and StreamMessage messages. Other kinds of JMS message are supported by other domains. For further information about using JMS messages with WebSphere Message Broker, see WebSphere Broker JMS Transport.

Manipulating messages in the IDOC domain

Use ESQL from a Compute node to copy the incoming IDoc to the outgoing IDoc, and manipulate the message.

A valid IDoc message flows out of SAP and is sent to the MQSeries link for R/3.

When this IDoc has been committed successfully to the outbound WebSphere MQ queue, the input node of the message flow reads it from that queue and generates the syntax element tree.

The Compute node manipulates this syntax element tree and, when it has finished, passes the output message to subsequent nodes in the message flow. When the message reaches the output node, the IDOC parser is called to rebuild the bit stream from the tree.

The message flow must create an output message in a similar format to the input message.

See Field names of the IDOC parser structures for the field names in the DC (Control Structure) and DD (Data Structure) that are recognized by the IDOC parser

Use the following ESQL example from a Compute node:

```
SET OutputRoot = InputRoot;
SET OutputRoot.IDOC.DC[1].tabnam = 'EDI_DC40  ';
SET OutputRoot.IDOC.DD[2].sdatatag.MRM.maktx = 'Buzzing all day';
```

The first line of the code copies the incoming IDoc to the outgoing IDoc.

The second line sets the *tabname* of the first DC.

The third line uses the second DD segment, which in this example is of type `E2MAKTM001`, and sets the *maktx* field.

Accessing fields of the IDoc using ESQL:

Use the ESQL editor Content Assist to fill in the SAP-defined fields of the IDoc.

After the `sdatatag` tag in an ESQL statement, the next tag is `MRM`, which you must enter manually, followed by the field name that is to be manipulated. Specify the name of the field within the message segment here, not the name of the message segment.

For example, the following code sets the segment name of the IDoc:

```
SET OutputRoot.IDOC.DD[I].segnam  = 'E2MAKTM001';
```

The following example sets the `msgfn` field within the `E2MAKTM001` segment:

```
SET OutputRoot.IDOC.DD[I].sdatatag.MRM.msgfn = '006';
```

Manipulating messages in the MIME domain

A MIME message does not need to be received over a particular transport. For example, a message can be received over HTTP by using an HTTPInput node, or over WebSphere MQ by using an MQInput node. The MIME parser is used to process a message if the message domain is set to MIME in the input node properties, or if you are using WebSphere MQ, and the MQRFH2 header has a message domain of MIME.

This topic explains how to deal with messages that belong to the MIME domain, and are parsed by the MIME parser. Use this information in conjunction with the information in “Manipulating message body content” on page 327.

You can manipulate the logical tree using ESQL before passing the message on to other nodes in the message flow. A message flow can also create a MIME domain tree using ESQL. When a MIME domain message reaches an output node, the MIME parser is called to rebuild the bit stream from the logical tree.

The following examples show how to manipulate MIME messages:
- "Creating a new MIME tree"
- "Modifying an existing MIME tree" on page 451
- "Managing Content-Type" on page 451

Creating a new MIME tree

A message flow often receives, modifies, and returns a MIME message. In this case, you can work with the valid MIME tree that is created when the input message is parsed. If a message flow receives input from another domain, such as XMLNS, and returns a MIME message, you must create a valid MIME tree. Use the following ESQL example in a Compute node to create the top-level structure for a single-part MIME tree:

```
CREATE FIELD OutputRoot.MIME TYPE Name;
DECLARE M REFERENCE TO OutputRoot.MIME;
CREATE LASTCHILD OF M TYPE Name NAME 'Data';
```

The message flow must also ensure that the MIME Content-Type is set correctly, as explained in "Managing Content-Type" on page 451. The flow must then add the message data into the MIME tree. The following ESQL examples show how you can do this. In each case, a Data element is created with the domain BLOB.

- A bit stream from another part of the tree is used. This example shows how a bit stream could be created from an XML message that is received by the message flow. The flow then invokes the BLOB parser to store the data under the Data element.

  ```
  DECLARE partData BLOB ASBITSTREAM(InputRoot.XMLNS);
  CREATE LASTCHILD OF M.Data DOMAIN('BLOB') PARSE(partData);
  ```

- Instead of parsing the bit stream, create the new structure, then attach the data to it, as shown in this ESQL example:

  ```
  DECLARE partData BLOB ASBITSTREAM(InputRoot.XMLNS);
  CREATE LASTCHILD OF M.Data DOMAIN('BLOB') NAME 'BLOB';
  CREATE LASTCHILD OF M.Data.BLOB NAME 'BLOB' VALUE partData;
  ```

Both of these approaches create the same tree structure. The first approach is better because explicit knowledge of the tree structure that the BLOB parser requires is not built into the flow.

More commonly, the Compute node must build a tree for a multipart MIME document. The following ESQL example shows how you can do this, including setting the top-level Content-Type property.

```
DECLARE part1Data BLOB ASBITSTREAM(InputRoot.XMLNS, InputProperties.Encoding, InputProperties.CodedCharSetId);

SET OutputRoot.Properties.ContentType = 'multipart/related; boundary=myBoundary';

CREATE FIELD OutputRoot.MIME TYPE Name;
DECLARE M REFERENCE TO OutputRoot.MIME;
CREATE LASTCHILD OF M TYPE Name NAME 'Parts';
CREATE LASTCHILD OF M.Parts TYPE Name NAME 'Part';
DECLARE P1 REFERENCE TO M.Parts.Part[1];
CREATE FIELD P1."Content-Type" TYPE NameValue VALUE 'text/plain';
CREATE FIELD P1."Content-Id"   TYPE NameValue VALUE 'part one';
CREATE LASTCHILD OF P1 TYPE Name NAME 'Data';
CREATE LASTCHILD OF P1.Data DOMAIN('BLOB') PARSE(part1Data);

CREATE LASTCHILD OF M.Parts TYPE Name NAME 'Part';
DECLARE P2 REFERENCE TO M.Parts.Part[2];
CREATE FIELD P2."Content-Type" TYPE NameValue VALUE 'text/plain';
CREATE FIELD P2."Content-Id"   TYPE NameValue VALUE 'part two';
CREATE LASTCHILD OF P2 TYPE Name NAME 'Data';
CREATE LASTCHILD OF P2.Data DOMAIN('BLOB') PARSE(part2Data);
```

Modifying an existing MIME tree

This ESQL example adds a new MIME part to an existing multipart MIME message. If the message is not multipart, it is not modified.

```
SET OutputRoot = InputRoot;

-- Check to see if the MIME message is multipart or not.
IF LOWER(InputProperties.ContentType) LIKE 'multipart/%'
THEN
  CREATE LASTCHILD OF OutputRoot.MIME.Parts NAME 'Part';

  DECLARE P REFERENCE TO OutputRoot.MIME.Parts.[<];
  CREATE FIELD P."Content-Type" TYPE NameValue VALUE 'text/xml';
  CREATE FIELD P."Content-ID"   TYPE NameValue VALUE 'new part';
  CREATE LASTCHILD OF P TYPE Name NAME 'Data';

  -- This is an artificial way of creating some BLOB data.
  DECLARE newBlob BLOB '4f6e652074776f2074687265650d0a';
  CREATE LASTCHILD OF P.Data DOMAIN('BLOB') PARSE(newBlob);
END IF;
```

Managing Content-Type

When you create a new MIME message tree, or when you modify the value of the MIME boundary string, make sure that the MIME Content-Type header is set correctly by setting the ContentType value in the broker Properties subtree. The following example shows how to set the ContentType value for a MIME part with simple content:

```
SET OutputRoot.Properties.ContentType = 'text/plain';
```

Do not set the Content-Type value directly in the MIME tree or HTTP trees because the value is ignored or used inconsistently.

Manipulating messages in the BLOB domain

How to deal with messages that belong to the BLOB domain, and that are parsed by the BLOB parser.

You cannot manipulate the contents of a BLOB message, because it has no predefined structure. However, you can refer to its contents using its known position within the bit stream, and process the message with a minimum of knowledge about its contents.

The BLOB message body parser does not create a tree structure in the same way that other message body parsers do. It has a root element BLOB, that has a child element, also called BLOB, that contains the data.

You can refer to message content using substrings if you know the location of a particular piece of information within the BLOB data. For example, the following expression identifies the tenth byte of the message body:

```
InputBody.BLOB.BLOB[10]
```

The following expression references 10 bytes of the message data starting at offset 10:

```
SUBSTRING(InputBody.BLOB.BLOB from 10 for 10)
```

You can use the Mapping node to map to and from a predefined BLOB message, and to map to and from items of BLOB data.

Simple example to write a string in the output message:

The following simple example allows you to write some character data in your ESQL (for example, if you have read some character fields from a database) out as a BLOB:

```
CALL CopyMessageHeaders();
 -- CALL CopyEntireMessage();
 DECLARE mystring CHARACTER;
 SET mystring='hello';
 SET OutputRoot.BLOB.BLOB=CAST (mystring AS BLOB CCSID 1208);
```

Using the CALL statement to call a user-written routine

The ESQL CALL statement calls routines that have been created and implemented in different ways.

A routine is a user-defined function or procedure that has been defined by one of the following statements:

- CREATE FUNCTION
- CREATE PROCEDURE

You can use the CALL statement to call a routine that has been implemented in any of the following ways:

- ESQL
- Java
- As a stored procedure in a database
- As a built-in (broker-provided) function

  You can use CALL to call built-in (broker-provided) functions and user-defined SQL functions, but typically you would use their names directly in expressions .

For details of the syntax and parameters of the CALL statement, see CALL statement. For an example of the use of CALL, see the examples in CREATE PROCEDURE statement.

Calling an ESQL routine:

A routine is called as an ESQL method if the routine's definition specifies a LANGUAGE clause of ESQL or if the routine is a built-in function. An exact one-to-one matching of the data types and directions of each parameter, between the definition and the CALL, is required. An ESQL routine is allowed to return any ESQL data type, excluding List and Row.

Calling a Java routine:

A routine is called as a Java method if the routine's definition specifies a LANGUAGE clause of JAVA. An exact one-to-one matching of the data types and directions of each parameter, between the definition and the CALL, is required. If the Java method has a void return type, the INTO clause cannot be used because no value exists to return.

A Java routine can return any data type in the ESQL-to-Java data-type mapping table, excluding List and Row.

Calling a database stored procedure:

A routine is called as a database stored procedure if the routine's definition has a LANGUAGE clause of DATABASE.

When a call is made to a database stored procedure, the broker searches for a definition (created by a CREATE PROCEDURE statement) that matches the procedure's local name. The broker then uses the following sequence to resolve the name by which the procedure is known in the database and the database schema to which it belongs:

1. If the CALL statement specifies an IN clause, the name of the data source, the database schema, or both, is taken from the IN clause.
2. If the name of the data source is not provided by an IN clause on the CALL statement, it is taken from the DATASOURCE attribute of the node.
3. If the database schema is not provided by an IN clause on the CALL statement, but is specified on the EXTERNAL NAME clause of the CREATE PROCEDURE statement, it is taken from the EXTERNAL NAME clause.
4. If no database schema is specified on the EXTERNAL NAME clause of the CREATE PROCEDURE statement, the database's user name is used as the schema name. If a matching procedure is found, the routine is called.

The chief use of the CALL statement's IN clause is that it allows the data source, the database schema, or both, to be chosen dynamically at run time. (The EXTERNAL SCHEMA clause also allows the database schema which contains the stored procedure to be chosen dynamically, but it is not as flexible as the IN clause and is retained only for compatibility with earlier versions. Its use in new applications is deprecated.)

If the called routine has any DYNAMIC RESULT SETS specified in its definition, the number of expressions in the CALL statement's *ParameterList* must match the number of parameters to the routine, plus the number of DYNAMIC RESULT SETS. For example, if the routine has three parameters and two DYNAMIC RESULT SETS, the CALL statement must pass five parameters to the called routine. The parameters passed for the two DYNAMIC RESULT SETS must be list parameters; that is, they must be field references qualified with array brackets [ ]; for example, Environment.ResultSet1[].

A database stored procedure is allowed to return any ESQL data type, excluding Interval, List, and Row.

Accessing broker properties from ESQL

You can access broker properties, at run time, from the ESQL modules in your message flow nodes.

You can use broker properties on the right side of regular SET statements. For example:

```
DECLARE mybroker CHARACTER;
SET mybroker = BrokerName;
```

where `BrokerName` is the broker property that contains the name of the broker on which the message flow is running. However, you cannot use broker properties on the left side of SET statements. This restriction exists because, at run time, broker properties are constants: they cannot be modified, therefore their values cannot be changed by SET statements. If a program tries to change the value of a broker property, the error message `Cannot assign to a symbolic constant` is issued.

Broker properties:

- Are grouped by broker, execution group, flow, and node.
- Are case sensitive. Their names always start with an uppercase letter.
- Return NULL if they do not contain a value.

If your ESQL code already contains a variable with the same name as one of the broker properties, your variable takes precedence; that is, your variable masks the broker property. To access the broker property, use the form `SQL.<broker_property_name>`. For example: `SQL.BrokerName`.

Broker properties that are accessible from ESQL and Java shows the broker, flow, and node properties that are accessible from ESQL and indicates which properties are also accessible from Java.

Configuring a message flow at deployment time with user-defined properties

Use user-defined properties (UDPs) to configure message flows at deployment and run time, without modifying program code. You can give a UDP an initial value when you declare it in your program, or when you use the Message Flow editor to create or modify a message flow.

For an overview of user-defined properties, see “User-defined properties” on page 134.

See the DECLARE statement for an example of how to code a UDP statement.

In ESQL, you can define UDPs at the module or schema level.

After a UDP has been defined by the Message Flow editor, you can modify its value before you deploy it.

To configure UDPs:

1. Switch to the Broker Administration perspective or Broker Application Development perspective.

2. Double-click the broker archive (BAR) file in the Navigator view. The contents of the BAR file are shown in the Manage and Configure page of the Broker Archive editor.
3. Click the **Manage and Configure** tab. This tab shows the message flows in your broker archive; expand a flow to show the individual nodes that it contains.
4. Click the message flow that you are interested in. The UDPs that are defined in that message flow are displayed with their values in the **Properties** view.
5. If the value of the UDP is unsuitable for your current environment or task, change it to the value that you want. The value of the UDP is set at the flow level, and is the same for all eligible nodes that are contained in the flow. If a subflow includes a UDP that has the same name as a UDP in the main flow, the value of the UDP in the subflow is not changed.
6. Save your broker archive.

Now you are ready to deploy the message flow. See Deploying a broker archive file.

Using PHP

You can use the PHP scripting language for message routing and transformation.

Before you start:

Ensure that the PHP capability is enabled. To enable broker capabilities, use the **-f** parameter of the mqsichangebroker command.

Support for the PHP scripting language is available on Windows only.

When you use the PHPCompute node, you can customize it to determine the exact processing that it provides. To tailor the behavior of each node, create a PHP file that provides the processing that you require. You can edit your PHP files by using a text editor such as the default Eclipse text editor.

The PHP API simplifies tasks that involve message routing and transformation tasks. These tasks include accessing named elements in a message tree, setting their values, and creating elements, without the need to navigate the message tree explicitly.

This section provides the following information about developing PHP:

PHP overview

WebSphere Message Broker provides support for the PHP scripting language (on Windows only).

WebSphere Message Broker provides a PHPCompute node, which is a programmable node that supports message transformation and routing by using the PHP scripting language. For example:

```
$output_assembly->XMLNSC->doc->item = $input_assembly->MRM->structure->field;
```

This PHP code generates the following XML code:

```
<doc>
 <item>
  ... deep copy of field element from input tree
 </item>
</doc>
```

The PHPCompute node builds on this syntax to produce a powerful syntax for accessing WebSphere Message Broker trees.

For information about the PHP functions supported by WebSphere Message Broker, see PHP extensions.

For more information about the PHP scripting language, see the PHP: Hypertext Preprocessor Web site.

Creating PHP code for a PHPCompute node

Use these instructions to create your PHP code and associate it with your PHPCompute node.

To create your PHP code and associate it with a PHPCompute node, follow these steps:

1. Ensure that your PHP code is in a PHP script file, with an extension of `.php`:
   - If the required PHP code exists, import the PHP script file into the workspace (see Importing file systems into the workbench). Alternatively, ensure that the PHP script file is stored in the file system, so that the PHPCompute node can point to it directly.
   - If the required PHP code does not exist, create it by following the instructions in "Writing PHP code."
2. Associate the PHP code with a PHPCompute node in your message flow. Follow the instructions in "Associating PHP code with a PHPCompute node" on page 458.

Writing PHP code

Use these instructions to create your PHP code.

1. Ensure that the PHP capability is enabled. For information about enabling broker capabilities, see the mqsichangebroker command.
2. In the Broker Application Development perspective, click **File** > **New** > **Other**. The Select a wizard pane is displayed.
3. Select **File** from the list of resources, then click **Next**.
4. Select the required parent folder from the list, then type the name of your new PHP script file in the **File Name** field. Ensure that the file you specify has an extension of `.php` (for example, `Hello.php`). The Eclipse text editor opens, with an empty pane in which you can type your PHP code.

5. Type your PHP code into your new PHP script file, by using the Eclipse text editor. The PHP script must be contained within the <?php and ?> tags:

```
<?php

// Body of the script

?>
```

You can create a PHP script with or without a class and evaluate method. The option you choose affects both the content of the script and the setting of the PHPCompute node's Invoke evaluate method property:

- Create a script including a class and evaluate method:

  The Invoke evaluate method property of the PHPCompute node is selected by default, therefore a class and evaluate method are expected in the PHP script.

  The PHP code must contain a class with the same name as the PHP file (Hello, for example), and this class must contain a function called evaluate, with parameters for the input and output message assemblies:

```
<?php

class Hello {

  /**
   * An example of MessageBrokerSimpleTransform
   * @MessageBrokerSimpleTransform
   */
  function evaluate($output_assembly, $input_assembly) {
   // transformation code here
   // $output_assembly ->XMLNSC->... = $input_assembly->XMLNSC->...
  }

}

?>
```

  For more information about the @MessageBrokerSimpleTransform annotation shown in this example, see "Using annotations" on page 458.

- Create a script without a class and evaluate method:

  The global variable *$assembly* makes the incoming message assembly available to the script. The incoming message and message assembly are read only. As a result, for message transformation, you must make a new copy of the message and the assembly containing the new message:

```
<?php

$output_message = new MbsMessage();

// transformation code here
// $output_message->XMLNSC->... = $assembly->XMLNSC->...

$output_assembly = new MbsMessageAssembly($assembly, $output_message);
$output_assembly->propagate("out");

?>
```

  The Invoke evaluate method property of the PHPCompute node is selected by default, therefore a class and evaluate method are expected in the PHP script. If you use a PHP script without a class and evaluate method, remember to clear the **Invoke evaluate method** property of the PHPCompute node.

  You must explicitly propagate the message assembly to one of the output terminals before the end of the script.

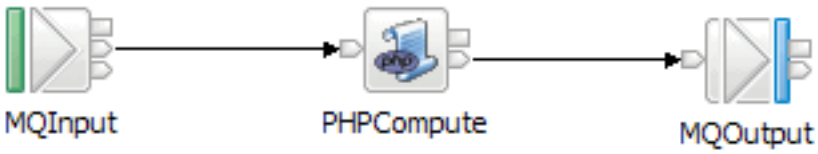
When you have created your PHP code, associate it with the PHPCompute node by following the instructions in "Associating PHP code with a PHPCompute node."

For information about the PHP scripting language, see the PHP: Hypertext Preprocessor Web site.

Associating PHP code with a PHPCompute node

Use these instructions to associate your PHP code with a PHPCompute node.

1. Create a message flow containing a PHPCompute node. For example, create a message flow containing an MQInput node, a PHPCompute node, and an MQOutput node. For information, see "Creating a message flow" on page 259.
2. Connect the Out terminal of the MQInput node to the In terminal of the PHPCompute node.
3. Connect the Out terminal of the PHPCompute node to the In terminal of the MQOutput node.



4. On the MQInput node, set the Queue name property (on the **Basic** tab) to InQueue.
5. On the PHPCompute node, set the PHP script property (on the **Basic** tab) to Hello.php. You can either type the name of the PHP file into the field, or select **Browse** to navigate to the PHP files stored in your workspace. If you select **Browse**, the PHP files in the current message flow project are displayed, together with PHP files in any referenced projects. To add project references, right-click the message flow project, select **Properties**, then select the **Project references** category.
6. Set the following properties of the MQOutput node:
   a. On the **Basic** tab, set the Queue name property to OutputQueue
   b. On the **Validation** tab, set the Validate property to Inherit.
7. Save the message flow.

You can display the PHP script file associated with the PHPCompute node by double-clicking the node in the message flow. The file `Hello.php` is opened in the Eclipse text editor.

Using annotations

Annotations alter the behavior of the evaluate method when using the PHP class structure, and remove the need to repeat commonly used code (for transforming message trees) in each PHP script.

When you use the PHP class structure with WebSphere Message Broker, the class must have the same name as the PHP file and it must implement a method called evaluate. The PHPCompute node instantiates the class and calls the evaluate method. For more information about developing PHP code, see "Creating PHP code for a PHPCompute node" on page 456.

The following annotations are supported by the broker:

- "@MessageBrokerSimpleTransform"
- "@MessageBrokerCopyTransform"
- "@MessageBrokerRouter" on page 460
- "@MessageBrokerLocalEnvironmentTransform" on page 461

You can specify multiple annotations for an evaluate method. If the MessageBrokerCopyTransform and MessageBrokerSimpleTransform annotations are specified together, the MessageBrokerCopyTransform annotation takes precedence. The input assembly is available with both the MessageBrokerSimpleTransform and MessageBrokerCopyTransform annotations.

If no annotations are specified, the first argument to the evaluate method is a read-only assembly. Annotation names are case-sensitive, and annotations that are not recognized are ignored.

When you use an annotation, the output assembly is passed to the evaluate method as the first parameter, and the input assembly is passed as the second parameter. The second parameter is optional and is passed in if you have specified it in your evaluate method declaration.

@MessageBrokerSimpleTransform:

Use the @MessageBrokerSimpleTransform annotation to alter the behavior of the evaluate method in a PHP class.

The @MessageBrokerSimpleTransform annotation causes two parameters to be passed to the evaluate method. The first parameter is a reference to the output assembly, and the second parameter is a reference to the input assembly. The second parameter is optional.

If the MessageBrokerSimpleTransform and MessageBrokerCopyTransform annotations are specified together, the MessageBrokerCopyTransform annotation takes precedence.

The following example copies the subtree under element *aaa* into the output tree under element *bbb*:

```
<?php

class SimpleTransform {

/**

     * An example of MessageBrokerSimpleTransform
     * @MessageBrokerSimpleTransform

     */

    function evaluate($output_assembly, $input_assembly) {

        $output_assembly->XMLNSC->bbb = $input_assembly->XMLNSC->aaa;
    }

}

?>
```

@MessageBrokerCopyTransform:

Use the @MessageBrokerCopyTransform annotation to alter the behavior of the evaluate method in a PHP class.

The @MessageBrokerCopyTransform annotation causes one parameter to be passed to the evaluate method. This parameter is a reference to the output assembly with the contents of the input assembly already copied into it. The input assembly is available with the @MessageBrokerCopyTransform. If you declare a second parameter (which is optional) the input assembly is passed to it.

If the @MessageBrokerCopy Transform and @MessageBrokerSimpleTransform annotations are specified together, the @MessageBrokerCopyTransform annotation takes precedence.

The following example modifies the original XML message by adding an element called *Greeting* with the value *Hello World*:

```
<?php

class CopyTest {

    /**

     * An example of MessageBrokerCopyTransform

     *

     * @MessageBrokerCopyTransform

     */

    function evaluate($assembly) {
        $assembly->XMLNSC->doc->Greeting = "Hello World";
    }

}

?>
```

@MessageBrokerRouter:

Use the @MessageBrokerRouter annotation to alter the behavior of the evaluate method in a PHP class.

The @MessageBrokerRouter annotation causes the return value of the evaluate method to be used to specify the terminal through which the message is to be propagated. The terminal can be either the Out terminal (defined on the node) or a dynamic terminal that you have created. You can add output terminals dynamically to your node instance in the Message Flow editor. The string that is returned from the evaluate method must match either the name of the dynamic terminal that you have defined or the Out terminal. If no return value is specified, the output assembly is not propagated to the next node after the evaluate method returns.

The following example routes the message to the Out terminal if the value of the *threshold* element is greater than 10; otherwise the message is routed to the *other* terminal:

```
<?php
class RouteTest {


    /**
```

```
     * Basic routing of a message.
     *
     * @MessageBrokerRouter
     */
    function evaluate($assembly) {
 // Simple filter
 if ($assembly->XMLNSC->doc->threshold->getValue() > 10) {
       return 'out';
 } else {
  return 'other';
  }
 }
}
?>
```

For information about dynamic terminals, see "Using dynamic terminals" on page 279.

@MessageBrokerLocalEnvironmentTransform:

Use the @MessageBrokerLocalEnvironmentTransform annotation to alter the behavior of the evaluate method in a PHP class.

The @MessageBrokerLocalEnvironmentTransform is similar to the @MessageBrokerSimpleTransform annotation but creates a copy of the local environment tree in the output assembly.

If the @MessageBrokerLocalEnvironmentTransform is used, nodes downstream of the PHPCompute node see changes to the local environment. If the @MessageBrokerLocalEnvironmentTransform is not used, the node can still modify the local environment, and all nodes in the flow (including upstream nodes) can see the changes.

The following example populates two new folders in the local environment tree, and copies the *Wildcard* subtree from the local environment into the output message:

```
<?php

class LocalEnvironmentTest {


    /**

     * Test local environment messages.

     *

     * @MessageBrokerSimpleTransform

     * @MessageBrokerLocalEnvironmentTransform

     */

    function evaluate($output_assembly, $input_assembly) {



        $output_assembly[MB_LOCAL_ENVIRONMENT]->Folder1 = 'some string';

        $output_assembly[MB_LOCAL_ENVIRONMENT]->Folder2->SubFolder =
          'another string';
```

```
        $output_assembly->XMLNSC->Message->InputLocalEnvironment =
          $input_assembly[MB_LOCAL_ENVIRONMENT]->Wildcard;

    }

}


?>
```

Using PHP arrays

PHP arrays are associative arrays (maps) but you can treat them as lists by adding values without keys.

PHP supports the following syntax for building arrays:

```
$array[] = "aaa";
$array[] = "bbb";
```

PHP allows an object to function as an array, and you can use this capability to create repeating elements in a tree. For example:

```
$output_root->XMLNSC->a->b->c[] = $input_root->XMLNSC->a->b;
$output_root->XMLNSC->a->b->c[] = $input_root->XMLNSC->a->c;
```

The code shown above creates the following elements:

```
<a>
 <b>
  <c>
   ... // contents of $input_root->XMLNSC->a->b
  </c>
  <c>
   ... // contents of $input_root->XMLNSC->a->c
  </c>
 </b>
</a>
```

You can use the array operator in the path, as shown in the following example:

```
$output_root->XMLNSC->a->b[]->c = $input_root->XMLNSC->a->b;
$output_root->XMLNSC->a->b[]->c = $input_root->XMLNSC->a->c;
```

to create the following elements:

```
<a>
 <b>
  <c>
   ... // contents of $input_root->XMLNSC->a->b
  </c>
 </b>
 <b>
  <c>
   ... // contents of $input_root->XMLNSC->a->c
  </c>
 </b>
</a>
```

The following example uses no array operators:

```
$output_root->XMLNSC->a->b->c = $input_root->XMLNSC->a->b;
$output_root->XMLNSC->a->b->c = $input_root->XMLNSC->a->c;
```

The example above produces the following XML code:

```
<a>
 <b>
  <c>
   ... // contents of $input_root->XMLNSC->a->c (overwriting previous)
  </c>
 </b>
</a>
```

You can also iterate a set of repeating elements by using a *foreach* loop, as shown in the following example:

```
foreach ($input_root->XMLNSC->doc->item as $item) {
   $output_root->XMLNSC->msg->bit[] = $this->transformItem($item);
}
```

This example builds an output message with a repeating *bit* element, one for each *item* element in the input message. The content of each *bit* element is determined by the *transformItem* function, which refers to the *item* element as its parameter. The following sample shows an example of this syntax being used for message transformation:

- PHPCompute Node

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

You can assign standard PHP arrays into an element tree as a way of building subtrees by using a very compact syntax. The following example shows how to build a repeating element from an array of values:

```
$output_root->XMLNSC->doc->item[] = array('aaa', 'bbb', 'ccc');
```

This code builds a tree with three item elements:

```
<doc>
 <item>aaa</item>
 <item>bbb</item>
 <item>ccc</item>
</doc>
```

Although the PHP array looks like a list, it is an associative array with the keys *0*, *1*, and *2*. The following example shows how to assign key/value pairs into the element tree:

```
$output_root->XMLNSC->doc->item = array('book'  => 'PHP',
                                        'fruit' => 'apple',
                                        'dog'   => 'Spaniel'   );
```

Without the [] operator on the item element, the keys in the array are used to name the child elements:

```
<doc>
 <item>
  <book>PHP</book>
  <fruit>apple</fruit>
  <dog>Spaniel</dog>
 </item>
</doc>
```

You can also nest arrays to represent more complex structures. For example:

```
output_root->XMLNSC->doc->items =
             array('book'  => array('title'  => 'PHP',
                                    'author' => 'A N Other'),
```

```
                'fruit' => 'apple',
                'dog'   => array('breed'  => 'Spaniel',
                                 'ears'   => 'long') );
```

The preceding example produces the following XML code:

```
<doc>
 <items>
  <book>
   <title>PHP</title>
   <author>A N Other</author>
  </book>
  <fruit>apple</fruit>
  <dog>
   <breed>Spaniel</breed>
   <ears>long</ears>
  </dog>
 </items>
</doc>
```

MbsElement arrays

When an array of MbsElement objects is assigned to an element tree, the array acts as an associative array.

For example:

```
$output_assembly->XMLNSC->doc->folder = $input_assembly->xpath("//item");
```

This example generates the following result:

```
<doc>
 <folder>
  <item>
   ... deep copy of 1st item element
  </item>
  <item>
   ... deep copy of 2nd item element
  </item>
  <item>
   ... deep copy of 3rd item element
  </item>
 </folder>
</doc>
```

The name of each element on the right side (in this example, *item*) becomes the name of the child element in the target tree. However, if the [] operator is used on the left side, *item* is replaced with *folder*, as shown in the following example:

```
$output_assembly->XMLNSC->doc->folder[] = $input_assembly->xpath("//item");
```

The example shown above generates the following result:

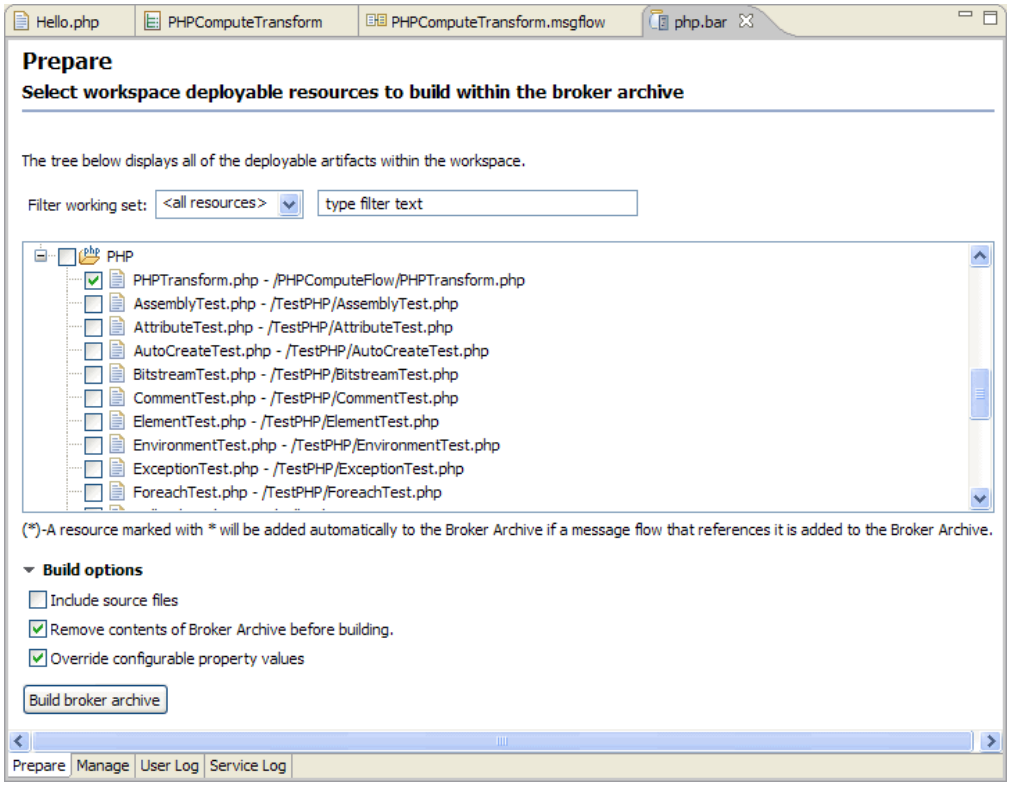
```
<doc>
 <folder>
  ... deep copy of 1st item element
 </folder>
 <folder>
  ... deep copy of 2nd item element
 </folder>
 <folder>
  ... deep copy of 3rd item element
 </folder>
</doc>
```

Deploying code in a PHPCompute node

The Message Broker Toolkit deploys the PHPCompute node code automatically.

When you create a broker archive (BAR) file and add the message flow, the Message Broker Toolkit packages the PHP code and its dependencies into the BAR file.

PHP files referenced by a PHPCompute node are added to the BAR file automatically when it is built, but you can also add additional PHP files to the BAR file by using the Broker Archive editor:



Accessing elements in the message tree from a PHPCompute node

Access the contents of a message, for reading or writing, by using the structure and arrangement of the elements in the tree that the parser creates from the input bit stream.

You can extract information from a message by using the PHPCompute node path syntax, or by using the MbsElement API methods. XPath 1.0 is supported as a means of accessing parts of the message.

Follow the relevant parent and child relationships from the top of the tree downwards until you reach the required element.

The following topics provide more information about accessing, extracting, and updating information in the message tree:

- "Traversing the element tree" on page 466
- "Accessing information about an element" on page 467

Traversing the element tree

Use PHP methods to access element trees.

Use the following statements to traverse a message tree from an element:

getParent()
Returns the parent of the current element.

getPreviousSibling()
Returns the previous sibling of the current element.

getNextSibling()
Returns the next sibling of the current element.

getChild()
Returns the first child of the current element, whose name is given by the first parameter. The n^{th} occurrence of that child element can be returned by specifying the second optional parameter.

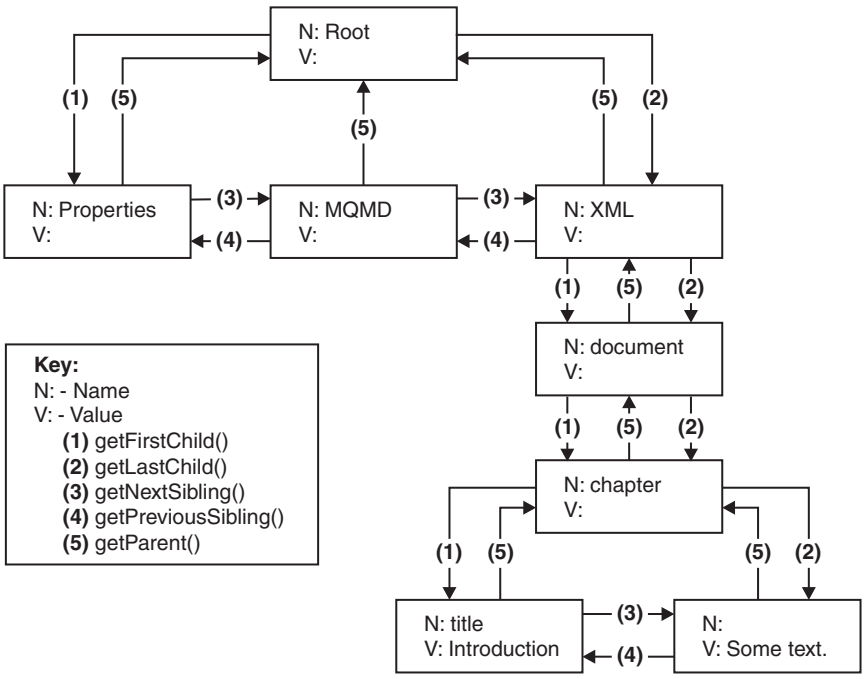
getChildren()
Returns all the child elements of the current element as an array of MbsElements. If the **namespace** parameter is specified, the array contains only the child elements with that namespace URI.

getFirstChild()
Returns the first child of the current element.

getLastChild()
Returns the last child of the current element.

The following example shows a simple XML message and the logical tree that is created from the message. The message has been sent by using WebSphere MQ. The logical tree diagram also shows the methods to call to navigate around the tree:

```
<document>
  <chapter title='Introduction'>
    Some text
  </chapter>
</document>
```



The tree used in this diagram is the one that is created by parsing the XML example given in this topic.

- From the Root part of the tree, calling getFirstChild() navigates to Properties. Also from Root, calling getLastChild() returns XML.
- From Properties, calling getParent() returns Root, and calling getNextSibling() returns MQMD.
- From MQMD, calling getPreviousSibling() returns Properties, calling getParent() returns Root, and calling getNextSibling() returns XML.
- From XML, calling getPreviousSibling() returns MQMD, calling getParent() returns Root, calling getFirstChild() returns document, and calling getLastChild() also returns document.
- From document, calling getParent() returns XML, calling getFirstChild() returns chapter, and calling getLastChild() also returns chapter.
- From chapter, calling getParent() returns document, calling getFirstChild() returns title, and calling getLastChild() returns the child that contains the message data "Some text.".

The following example shows how to use the MbsElement methods to navigate to the chapter element:

```
$chapter = $input_assembly->getLastChild()->getFirstChild()->getFirstChild();
```

The following example shows how to navigate to the chapter element by using the path syntax:

```
$chapter = $input_assembly->XML->document->chapter;
```

Accessing information about an element

Use PHP methods to return information about an element.

Use the following methods to return information about the referenced element:

getName()
Returns the name of the current element

getValue()
Returns the value of the current element

getType()
Returns the specific type of the current element

getNamespace()
Returns the namespace URI of the current element

Creating and transforming messages using a PHPCompute node

You can use the PHPCompute node to create and copy messages, and to create and manipulate message elements.

These topics describe how to transform messages by using a PHPCompute node:

- "Creating a new message using a PHPCompute node"
- "Copying a message using a PHPCompute node" on page 469
- "Setting and moving message elements using a PHPCompute node" on page 469
- "Creating new elements using a PHPCompute node" on page 470

Creating a new message using a PHPCompute node

Create a new output message by using the PHPCompute node.

When you use a PHPCompute node to create a new output message, the code required depends on whether the annotated evaluate method is defined in the PHP script.

If you use the @MessageBrokerSimpleTransform annotation, the new output message and message assembly objects are created automatically. For example:

```
<?php

class MyNode {

 /**
  * @MessageBrokerSimpleTransform
  */
 function evaluate($output_assembly, $input_assembly) {
  // $output_assembly refers to the new message
 }
}

?>
```

If you use a plain script without an annotated evaluate method, you must create the output message and message assembly objects explicitly:

```
<?php

// the output message must be created explicitly
$output_message = new MbsMessage();

// a new output message assembly must be created to hold this new message
$output_assembly = new MbsMessageAssembly($assembly, $output_message);

?>
```

Copying a message using a PHPCompute node

Copy an existing message using the PHPCompute node.

When you use a PHPCompute node to copy an existing message, the code required depends on whether or not the annotated evaluate method is defined in the PHP script.

If you use the @MessageBrokerCopyTransform annotation, the new output message and message assembly objects are created automatically. For example:

```
<?php

class MyNode {

 /**
  * @MessageBrokerCopyTransform
  */
 function evaluate($output_assembly, $input_assembly) {
  // $output_assembly refers to the new message
 }
}

?>
```

If you use a plain script without an annotated evaluate method, you must create the output message and message assembly objects explicitly:

```
<?php

// create a copy of the input message
$output_message = new MbsMessage($assembly[MB_MESSAGE]);

// a new output message assembly must be created to hold this new message
$output_assembly = new MbsMessageAssembly($assembly, $output_message);

?>
```

Setting and moving message elements using a PHPCompute node

You can transform elements in the message as it passes through a PHPCompute node in the message flow.

The following sections show the methods that you can use to modify, move, and remove elements:

- "Setting information about an element"
- "Moving elements" on page 470
- "Removing elements" on page 470

The PHP API reference information provides details about each of the methods used in the following sections.

Setting information about an element:

Use these methods to set information about the referenced element:

setName()
: Sets the name of the current element

setValue()
: Sets the value of the current element

setType()
Sets the specific type of the current element

setNamespace()
Sets the namespace URI of the current element

You can also set the value of an element by using the assignment operator. For example, `$element = 'text';` is equivalent to `$element.setValue('text');`.

Moving elements:

Use a PHPCompute node to copy or detach an element from a message tree by using the following methods:

detach()
Detaches the current element from the tree

detachAllChildren()
Detaches all children of the current element from the tree

Use one of the following methods to attach an element or subtree that you have copied on to another tree:

addElement(*element***)**
Adds an element as the last child (by default) of the current element

addAttribute(*attribute***)**
Adds an attribute to the current element

Removing elements:

Use these methods to remove elements from the message tree:

detach()
Detaches the current element from the tree

detachAllChildren()
Detaches all children of the current element from the tree

You can also remove elements from the message tree by using the PHP unset() function. For example:

```
unset($output_assembly->XMLNSC->doc->folder->item);
$output_assembly->XMLNSC->doc->folder->item->detach();
```

Creating new elements using a PHPCompute node

Use the PHPCompute node to create new elements in a message tree.

You can create new elements in a message tree in several ways:

- By using the MbsElement addElement method. By default this creates an element as the last child of the current element. This method has the following parameters:
  - **name**
  - **value**
  - **namespace**
  - **type** (optional)
  - **position** (optional)

  The **type** parameter is the parser-specific type of the new node, which defaults to the XML element type for XML parsers.

The **position** parameter can have one of the following values:
– *MB_FIRST_CHILD*
– *MB_LAST_CHILD*
– *MB_NEXT_SIBLING*
– *MB_PREVIOUS_SIBLING*

- By using the path syntax. Elements on the left side of an assignment expression are created (if they do not exist already) when they are referenced in a path. For example, the following code navigates the elements in the XMLNSC tree, creating them if necessary:

```
$output_assembly->XMLNSC->doc->folder->item = 'book';
```

- By using an MbsElement constructor. To create a PHP function that creates and returns a subtree (part of a message), you can instantiate an element, build extra elements (using either of the previous two methods described), and return the result. You can then assign the result into the output message. For example:

```
$output_assembly->XMLNSC->doc->part = create_subtree();

function create_subtree() {
 $element = new MbsElement();
 $element->folder->item = 'some value';
 return $element;
}
```

  You can also use the MbsElement addElementFromBitstream method to create an element tree from a string containing an unparsed bit stream. Use this method to defer until run time the decision about which parser to use.

XML support

The PHP capability in WebSphere Message Broker provides support for XML.

XML namespaces

The path navigation syntax in the PHPCompute node is not namespace aware. As a result, the expression shown in the following example navigates through the catalogue and entry elements regardless of the namespace URI of the elements:

```
$ref->catalogue->entry
```

If you generate an output message that requires namespace elements, set the namespace URI after you create the path:

```
$table->entry = $ref->catalogue->entry;
$table->entry->setNamespace('http://www.ibm.com/namespaceURI');
```

Alternatively, you can create the entry element by using the addElement API method:

```
$value = $ref->catalogue->entry;
$table->addElement('entry', $value, 'http://www.ibm.com/namespaceURI');
```

XML attributes

XML attributes are stored in the element tree as MbsElements with a *type* value that identifies them as attributes. The path syntax supports addressing an attribute of an element, by using the array operator with the attribute name as the key; therefore, attributes function as map arrays on the element. For example, the following code returns the *name* attribute of the **folder** element:

```
$attr = $input->XMLNSC->doc->folder['name']
```

You can create attributes in a similar way; for example:

```
$output->XMLNSC->doc->folder['name'] = 'PHP';
```

This example generates the following XML code:

```
<doc>
 <folder name='PHP'/>
</doc>
```

Routing a message using a PHPCompute node

Route a message by using the PHPCompute node as a filter node.

Before you start:

Add a "PHPCompute node" on page 1122 to your message flow.

By default, the output message assembly is propagated to the Out terminal after the evaluate method in the PHP script has been processed. However, the PHPCompute node also has dynamic output terminals, so that you can use it as a filter node by propagating a message to the appropriate terminal, based on the message content.

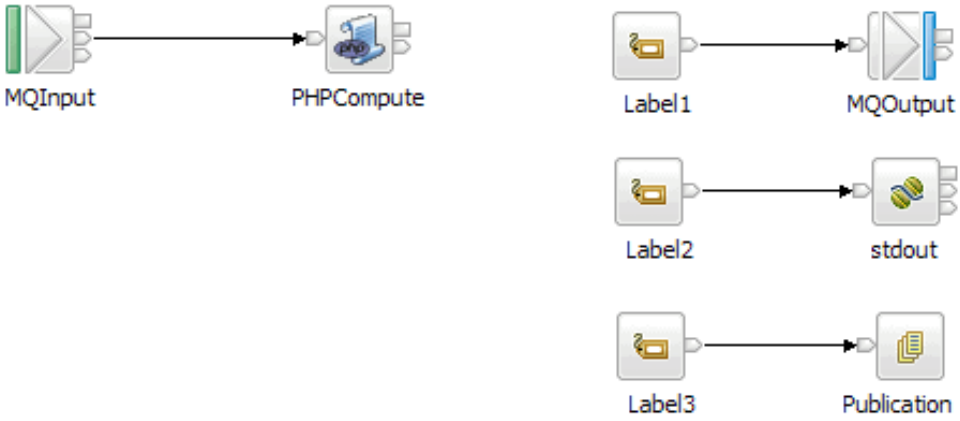
You can use the @MessageBrokerRouter annotation in your PHP code to route the message to a terminal specified by the string return value of the evaluate method. If no string is returned, the message is not propagated.

The following example shows the @MessageBrokerRouter annotation in a PHP script file:

```
/**
   * @MessageBrokerRouter
   */
  function evaluate($message) {
    if ($message->XMLNSC->doc->threshold->getValue() > 10) {
      return 'out';
    } else {
      return 'other';
    }
  }
```

For more information about using the @MessageBrokerRouter annotation for message routing, see "@MessageBrokerRouter" on page 460.

Alternatively, you can propagate a message directly to a Label node. When you use this method, you do not need to use a RouteToLabel node, and you do not need to propagate messages to output terminals.



The following example shows the PHP code associated with the PHPCompute node in the message flow shown above. The PHP code specifies that the message is to be routed to the node called *Label2*:

```
...
$output->routeToLabel('Label2');
```

Accessing other parts of the message tree using the PHPCompute node

You can use the PHPCompute node to access headers, broker properties, user-defined properties, the local environment, and the global environment.

The message tree is passed to a PHPCompute node as an argument of the evaluate method. The argument is an MbsMessageAssembly object. The message assembly contains four message objects:

- Parsed message
- LocalEnvironment
- GlobalEnvironment
- ExceptionList

The following constants are provided for accessing the different trees:

- MB_MESSAGE (default)
- MB_LOCAL_ENVIRONMENT
- MB_GLOBAL_ENVIRONMENT
- MB_EXCEPTION_LIST

The following topics describe how to access parts of the message tree:

- "Accessing headers with a PHPCompute node"
- "Updating the local environment with the PHPCompute node" on page 474
- "Updating the global environment with the PHPCompute node" on page 474
- "Accessing broker properties from the PHPCompute node" on page 475
- "Accessing user-defined properties from a PHPCompute node" on page 475

Accessing headers with a PHPCompute node

Access MQMD and MQRFH2 headers by using a PHPCompute node.

If an input node receives an input message that includes message headers that the input node recognizes, the node invokes the correct parser for each header. Parsers are supplied for most WebSphere MQ headers. This topic provides guidance for accessing the information in the MQMD and MQRFH2 headers that you can follow when accessing other headers that are present in your messages.

For more information about the contents of these and other WebSphere MQ headers for which WebSphere Message Broker provides a parser, see "Element definitions for message parsers" on page 1517.

Accessing the MQMD header:

WebSphere MQ and SCADA messages include an MQMD header. You can use a PHPCompute node to refer to the fields within the MQMD, and to update them.

The following PHP code shows how to add an MQMD header to your message:

```
$output_assembly->MQMD->Version = 2;
```

Accessing the MQRFH2 header:

The following PHP code adds an MQRFH2 header to an outgoing message that is to be used to make a subscription request:

```
$output_assembly->MQRFH2->psc->Topic = 'department/sales';
```

Updating the local environment with the PHPCompute node

The local environment tree is part of the logical message tree in which you can store information while the message flow processes the message.

Follow these steps to update the local environment:

1. Make a copy of the local environment to update.
2. Use the following PHP code to alter the copy of the local environment:

```
<?php

class LocalEnv {

  /**
   * @MessageBrokerLocalEnvironmentTransform
   */
  function evaluate($output_assembly, $input_assembly) {

    $output_assembly [MB_LOCAL_ENVIRONMENT] = $input_assembly->XMLNSC->Message;
    $output_assembly [MB_LOCAL_ENVIRONMENT]->Folder1 = 'some data';
    $output_assembly [MB_LOCAL_ENVIRONMENT]->Folder2->SubFolder = 'more data';
    }
}

?>
```

Updating the global environment with the PHPCompute node

Use PHP code associated with a PHPCompute to modify the global environment.

The Global Environment tree is always created when the logical tree is created for an input message. You can use this tree for your own purposes, for example to pass information from one node to another. You can use the whole tree as a scratchpad or working area.

The global environment can be altered across the message flow. The following PHP code shows how to alter the global environment:

```
<?php
class GlobalEnv {

  /**
   *    */
  function evaluate($output_assembly, $input_assembly) {

    $output_assembly [MB_GLOBAL_ENVIRONMENT] = $input_assembly->XMLNSC->Message;
    $output_assembly [MB_GLOBAL_ENVIRONMENT]->Folder1 = 'some data';
    $output_assembly [MB_GLOBAL_ENVIRONMENT]->Folder2->SubFolder = 'more data';
    }
}

?>
```

Accessing broker properties from the PHPCompute node

Access broker properties from the PHP code associated with a PHPCompute node.

For each broker, WebSphere Message Broker maintains a set of properties, and you can access some of these properties from your PHP code. You can have real-time access to details of a specific node, flow, or broker.

Four categories of broker property exist, which relate to:

- A specific node
- Nodes in general
- A message flow
- An execution group

Broker properties:

- Are grouped by broker, execution group, flow, and node
- Are case sensitive. Their names always start with an uppercase letter
- Return NULL if they do not contain a value

Broker properties are held in the $_ENV superglobal array. The following values are supported:

Broker property	**PHP variable**
Work Path	*$_ENV['MB_WORK_PATH']*
Broker Name (label)	*$_ENV['MB_BROKER_NAME']*
Execution Group Name (label)	*$_ENV['MB_EXECUTION_GROUP_NAME']*
Node Name (Label)	*$_ENV['MB_NODE_NAME']*
Message Flow Name (label)	*$_ENV['MB_MESSAGE_FLOW_NAME']*

Accessing user-defined properties from a PHPCompute node

Customize a PHPCompute node to access properties that you have associated with the message flow in which the node is included.

To access these properties from a PHPCompute node, use the mb_get_user_defined_property(*name*) method, where *name* is the name of the property that you are accessing. This method returns a PHP datatype equivalent to the ESQL broker type in the property definition.

For more information about the ESQL to PHP mappings, see PHP data types.

You can use the Configuration Manager Proxy (CMP) to change the value of user-defined properties. Use the getUserDefinedPropertyNames(), getUserDefinedProperty(), and setUserDefinedProperty() methods to query, discover, and set user-defined properties, as described in Setting user-defined properties at run time in a CMP application.

Calling Java from PHP

The IBM sMash Runtime for PHP provides access to Java classes and functionality from PHP. This Java Bridge can instantiate Java classes and call their methods.

You can manipulate values in an MRM or XMLNSC tree with xsd types that do not map directly to PHP types:

- xsd:decimal type java.math.BigDecimal
- xsd:dateTime com.ibm.broker.plugin.MbTimestamp

For example:

```
/**
* @MessageBrokerSimpleTransform
*/
function evaluate ($output, $input) {
 $number = $input->XMLNSC-doc->number->getValue();
 $signature = new JavaSignature (JAVA_STRING);
 $decimal = new Java("java.math.BigDecimal", $signature, "654.321");
 $sum = $number->add($decimal);
 $output->XMLNSC->doc->number = $sum;

 $timestamp = $input->XMLNSC->doc->date->getValue();
 $now = new Java("java.util.Date");
 $timestamp->setTime($now);
 $output->XMLNSC->doc->date = $timestamp;
}
```

Using XPath

XPath provides an alternative method to ESQL for entering expressions in the property fields of specific built-in nodes.

In addition to ESQL as a message transformation language, WebSphere Message Broker supports an alternative expression grammar in property fields. For more information, see ESQL-to-XPath mapping table.

You can use ESQL or XPath expressions in built-in nodes, within your message flows, to query or update message trees that are specified as accessible, and that you expect to be processed by a given node.

This section provides information on:

XPath overview

The XML Path Language (XPath) is used to uniquely identify or address parts of an XML document. An XPath expression can be used to search through an XML

document, and extract information from any part of the document, such as an element or attribute (referred to as a *node* in XML) in it. XPath can be used alone or in conjunction with XSLT.

Some of the built-in nodes provided in the workbench can use XPath expressions to specify the part of a message that is processed by the node. For example, you can use an XPath expression to identify fields in a message and determine if they match a specified value, or to set the field value by updating it with the results of a database query.

You can use XPath 1.0 path expressions in your flow to access specific parts of an incoming message, create or locate parts of an outgoing message, and perform complex message processing that might involve values present in message trees accessible by a node so that you can transform, filter, or retrieve values from a message.

For example, the Route node applies XPath 1.0 general expressions to the content of message trees associated with the incoming message assembly for this node. Following evaluation of an expression the result is cast as a Boolean (true or false) result, and this in turn is used to determine if a copy of the incoming message is routed down an output terminal associated with the processed expression.

If you have XML schema definition (`.mxsd`) files present in your workspace, any elements, attributes or data types defined in such definitions can be loaded into the Data types viewer and selected to automatically generate a path expressions mapping to the definition concerned.

Equally, depending on the XPath expressions supported by the property concerned, you can select XPath functions and operators to include in an expression, or you can build your own expressions manually.

The Data types viewer contains a list of variables relating to trees that can be accessed by expressions for the node property concerned.

For example, `$InputRoot` gives access to the incoming message tree. The fixed format of standard folders you can expect to exist under this tree, for example, `Properties` and `MQMD` are described without the need to import an `.mxsd` definition for them. These structures can be navigated in the viewer and, on selection of an element within them, a path expression that maps to the element in question is built automatically through the XPath 1.0 language.

For further information on XPath 1.0 see W3C XPath 1.0 Specification.

You can use the XPath Expression Builder to visually build XPath expressions to set the relevant properties in your nodes. You launch the XPath Expression Builder from buttons located along side property fields present in the Properties viewer, for those nodes that support the use of XPath expressions as property values.

The XPath files in WebSphere Message Broker are supplied in three property editors; see XPath property editors for more details.

The XPath editor supports both content-assist directly on the text field and also an **Edit...** button that launches the XPath builder dialog. The dialog gives you a larger area in which to build your XPath expressions.

The content assist based control contains two different proposal lists in the following order:

1. Nodes and Variables
2. Functions and Operators

The node and variable proposals are displayed the first time that you use the XPath editor. In this view, the status bar reads `Press **Ctrl+Space** to show Function and Operation Proposals`.

Pressing **Ctrl+Space** when you are in the function and operator level proposals selects the Node and Variable proposals.

Namespace support

The XPath Expression builder provides qualified support for namespaces.

The XPath Expression builder dialog contains a namespace settings table that:

- Supports simplified expressions that enable qualified namespace matching at run time
- Can be auto-generated based on imported schema definitions and generated expressions (based on selections made in the dialog when you build an expression)
- Allows you to add your own entries to the table

The table encapsulates deployable data passed to the runtime environment, as part of the nodes attribute data, and is used by the node to modify expressions through prefix-to-URI substitution. The final expressions support namespace matching, because they are processed against a target tree when employed by their associated message processing engine, that is, the XPath 1.0 runtime engine or ESQL runtime engine.

When you enter an ESQL field reference expression in a read-only or read-write path field, or an XPath 1.0 path expression in a read-only or read-write path field, or a general expression field (general expressions can contain zero or more path expressions), WebSphere Message Broker understands the language from the syntax you use.

XPath is the default for general expression fields that are validated by ensuring they conform to the XPath 1.0 grammar. For path expression fields XPath is assumed if the expression is valid and begins with a $ sign.

The language you can use is dictated by the property editor currently in use for a node's property field.

Namespace prefixes are used in an XPath or ESQL expression to make the statements shorter and easier to understand, while still supporting the ability to qualify an element name match by also matching on its associated namespace URI.

For example, consider the following message, where namespace prefix `b` is overridden through an inner declaration:

```
<b:a xmlns:b='xyz'>
 <!-- the namespace of elements 'a' and 'c' using prefix 'b'  is xyz -->
 <b:c>
  <b:d  xmlns:b='qrs'>
   <!-- the namespace of elements 'd' and 'e' using prefix 'b' is now qrs -->
```

```
    <b:e>100</b:e>
   </b:d>
  </b:c>
 </b:a>
```

Note that the scope of a namespace declaration declaring a prefix extends from the beginning of the start tag in which it appears to the end of the corresponding end tag, excluding the scope of any inner declarations with the same namespace prefix. In the case of an empty tag, the scope is the tag itself: >.

To navigate to element e in the above message use the following human-readable XPath expression:

```
/b:a/b:c/b2:d/b2:e
```

Note, that to prevent the auto-generated prefix to the URI map produced in the expression dialog overloading the same prefix (in this case b), the inner b prefix is appended with a numeric value to distinguish it from the outer b prefix. This strategy is repeated for each prefix name clash.

This notation is similar to the equivalent human-readable ESQL expression:

```
Root.b:a.b:c.b2:d.b2:e
```

To support namespace prefixes within expressions, the XPath Expression Builder Dialog automatically generates a prefix to a URI namespace settings table (based on the content of imported schema definitions, through which expressions are generated) .

Without the use of namespace prefixes to URI mapping data in this table, the runtime environment would be forced to take a less efficient approach, where portable but verbose XPath expressions would be required by it to provide namespace matching support.

The previous expression:

```
/b:a/b:c/b2:d/b2:e
```

would take the form:

```
/*[namespace-uri()='xyz' and local-name()='a']/*[namespace-uri()='xyz'
   and local-name()='c']/*[namespace-uri()='qrs' and
   local-name()='d']/*[namespace-uri()='qrs' and local-name()='e']
```

XPath Expression Builder

You can launch the XPath Expression Builder from most property fields that support, or expect, XPath expressions as a value that can be entered into the field.

The use of the XPath Expression Builder is optional, in that it is an aid to you in developing message flow applications. The XPath Expression Builder helps you to construct message processing expressions in either XPath or ESQL. You are free to enter expressions by hand, or use the XPath Expression Builder to help construct such expressions.

You can populate the fields, regardless of the state of the node; that is whether the node is detached or connected, or fully, partially, or completely unconfigured.

You launch the XPath Expression Builder from buttons that are within:

- Table cells, located to the right of the text entry field within the cell.

- **Add** or **Edit** dialogs used to construct rows within tables, located to right of the property field concerned.
- Tabs in the property viewer for a node, to the right of a property field.

Variables (or in ESQL terminology, correlation names) provide a list of all message tree start points that are applicable to the property field from which the dialog was launched.

If a field is a read-only or a read-write path field, expressions must start with such a variable to indicate which tree in which message assembly the path expression is mapping to.

XPath variable names map to existing correlation names found in ESQL field reference expressions, but to conform to the ESQL grammar they are designated as variable references by prefixing them with the dollar ($) character.

For example:

ESQL `Root.XMLNSC.CUST_DETAILS.NAME`

XPATH

```
$Root/XMLNSC/CUST_DETAILS/NAME
```

The variable indicates to which tree and where in that tree the expression is anchored.

The XPath Expression Builder dialog supports validation, which you can turn off on the XPath preferences page by clearing the **Validate when creating XPath expressions** check box.

If you select variables `$Root` or `$Body` and create an expression that refers to the body of the message, the XPath expression contains the message element. This is correct for message bodies owned by the XMLNSC, XMLNS, XML and DataObject domains.

For message bodies that are owned by the MRM, MIME, SOAP, and IDOC domains, you must remove the message element from the expression.

For example, the XPath expression `$Body/my_message/my_field` is correct for XMLNSC, but must be changed to `$Body/my_field` to be correct for MRM.

Views

There are three main views when functions are supported.

Whether a view is displayed, and what is displayed in it, depends on what type of property editor you have used to launch the dialog, and its tailored settings; for example, for path type fields you do not see a functions pane. The operators that are supported can change as can the list of applicable variables.

Data Types Viewer
This view shows the different schema types, elements, and attributes that you can use within the XPath expression that you are creating, as well as the allowable variable references.

XPath Function
This view shows four main top level categories, which are:

String This category corresponds to the description in the XPath 1.0 specification of `section-String-Functions`.

Boolean
This category corresponds to the description in the XPath 1.0 specification of `section-Boolean-Functions`.

Numeric
This category corresponds to the description in the XPath 1.0 specification of `section-Number-Functions`.

Nodeset
This category corresponds to the description in the XPath 1.0 specification of `section-Node-Set-Functions`.

For information on the format of XPath 1.0 expressions see the W3C XPath 1.0 Specification.

Operators
This view shows a list of all of the available operators that you can use within the given XPath expression.

Namespace settings

If you expand **Namespace settings** in the XPath Expression Builder dialog you see a table of Prefix and Namespace pair strings. This table is automatically updated when XPath expressions are created. If the default prefix generated is not what you want, you can change it by clicking **Change Prefix**.

To add a prefix and namespace map entry click **Add** and complete the fields in the dialog.

To edit or delete an entry in the table, select the item and click **Edit** or **Delete** respectively.

Edit opens another field dialog allowing you to change the prefix and namespace.

For information about the preferences supplied with the XPath editor, see "XPath editor preferences."

XPath editor preferences

This topic describes the possible options available to you when you use the XPath editor.

The following lists describe the custom preferences used in the XPath editor.

- Validation option:

  Validation when building XPath expressions
  Default: checked.

  This option is used to perform validation each time you update the XPath expression. As validation requires re-parsing the expression against the XML Schema document you can turn this option off, for example, if you are dealing with very large complex XPath expressions.

- Content Assist display options:

  Show XML Schema model groups
  Default: not checked.

  This option allows you to view XML schema model groups, or not.

Show type in XML Schema tree
Default: checked.

This option allows you to view the `<type name>` in both the Content Assist view and the XPath expression builder, or not.

Show function parameters
Default: checked.

This option allows you to have function parameters shown, or not.

Show function return type
Default: checked.

This option allows you to have function return types shown, or not.

Show content assist description
Default: checked.

This option allows you to view the description of a given selected entry in the Content Assist view, or not..

- Auto-Activation option:

  Enable auto activation
  Default: checked.

  When this option is active, the Content Assist field appears whenever the cursor is after one of the following:

  – / - Forward slash character
  – [- left bracket character
  – (- Left parentheses character
  – , - Comma character

  You set the delay time, before the Content Assist field appears, in the **Auto activation delay** field. The time is in milliseconds and the range is a positive number between zero and 9999.

- Content assist color options:

  This preference allows you to customize the background and foreground colors for the Content Assist fields. The default background color is (red, green, blue - 254, 241, 233) and the default foreground color is (red, green, blue - 0, 0, 0) - black.

XPath expressions supported by default

XPath supports the following expressions by default:

Read-only fields

```
$Root, $Body, $Properties, $LocalEnvironment,  $DestinationList,
$ExceptionList, $InputRoot , $InputBody, $InputProperties,
$InputLocalEnvironment, $InputDestinationList,
$InputExceptionList,$Environment.
```

To exclude variables for a node property from the default list, specify a comma separated string of variables. For example, specifying:

```
"com.ibm.etools.mft.ibmnodes.editors.xpath.XPathReadOnlyPropertyEditor", "InputRoot ,
 InputBody, InputProperties, InputLocalEnvironment, InputDestinationList,
 InputExceptionList"
```

restricts the XPath field to support only:

```
$Root, $Body, $Properties, $LocalEnvironment,  $DestinationList, $ExceptionList'
$Environment
```

Read-write fields

```
$InputRoot , $InputBody, $InputProperties, $InputLocalEnvironment,
$InputDestinationList, $InputExceptionList,  $OutputRoot ,
$OutputLocalEnvironment, $OutputDestinationList,
$OutputExceptionList, $Environment.
```

To exclude variables for a node property from the default list, specify a comma separated string of variables. For example, specifying:

```
"com.ibm.etools.mft.ibmnodes.editors.xpath.XPathReadWritePropertyEditor", "InputRoot ,
 InputBody, InputProperties, InputLocalEnvironment, InputDestinationList,
 InputExceptionList"
```

restricts the XPath field to support only:

```
$OutputRoot, $OutputLocalEnvironment, $OutputDestinationList, $OutputExceptionList,
 $Environment
```

Expression fields

```
$Root, $Body, $Properties, $LocalEnvironment,  $DestinationList,
$ExceptionList, $InputRoot , $InputBody, $InputProperties,
$InputLocalEnvironment, $InputDestinationList, $InputExceptionList,
$OutputRoot , $OutputLocalEnvironment, $OutputDestinationList,
$OutputExceptionList, $Environment.
```

To exclude variables for a node property from the default list, specify a comma separated string of variables. For example, specifying:

```
"com.ibm.etools.mft.ibmnodes.editors.xpath.XPathPropertyEditor", "InputRoot ,
InputBody, InputProperties, InputLocalEnvironment, InputDestinationList,
InputExceptionList, OutputRoot , OutputLocalEnvironment, OutputDestinationList,
OutputExceptionList"
```

restricts the XPath field to support only:

```
$Root, $Body, $Properties, $LocalEnvironment,  $DestinationList, $ExceptionList,
 $Environment.
```

Creating XPath expressions

A number of built-in primitive nodes have properties that can be specified using an XPath 1.0 expression; most commonly where this language is used to form a path expression to locate incoming message body elements received by a node.

Other less common node property fields support the entry of general XPath 1.0 expressions that support a wider range of the language to perform more complex evaluations in the broker's XPath 1.0 runtime engine.

The XPath Expression builder provides a tree view of a message, and supports the automatic generation of an XPath 1.0 path expression, through the selection of an element within the tree.

The Schema Viewer section provides a tree view of the input message. To visually build your XPath expression follow these steps:

1. Add the relevant node to your message flow
2. In the Properties viewer, enter the correlation name, or press Ctrl + Space to use content assist, or press Edit to use the Expression editor. Content assist is also invoked by simply typing $ in cell-based property fields. See "Correlation names" on page 88 for further information on correlation names.
3. Expand the tree, navigate to the field for which you want to build an expression, and click to select it. A field is either an element, or an attribute. Double click the field to add it to the XPath expression. You can also drag

fields, functions, and operators to the desired location in the XPath expression when using the XPath Expression builder.

4. To set conditions, enter them as you would a normal XPath Expression.

The complete XPath expression is shown either:

- In the XPath Expression pane if you are using the XPath Expression builder.

  The Expression builder dialog is an optional aid for generating expressions that, when complete, form the value in a node's property field.

  If you do not use the Expression builder dialog, the expressions entered manually are validated using the property editor.
- In the Property field if it is in the node itself.

Messages are displayed at the top of the XPath Editor window to alert you to the fact that a path or expression you have entered is not valid.

Note: The editor does not prevent you from entering, and saving, an expression that is not valid.

Here is the XPath expression built in the XPath Expression Builder to filter the Employee business objects for all employees who are managers: `$Body/getEmployeeInfo/Emp[isManager=true()]`.

- `$Body`: The body section of the message, that is, the last child of root.
- `/getEmployeeInfo`: The name of the operation in the interface.
- `/Emp`: The name of the input message type.
- `[isManager=true()]`: Checks whether the `isManager` field is set to true.

In this case the same expression works for both request and response flows, because the input and output messages for the operation are identical.

See W3C XPath 1.0 Specification for more information on XPath 1.0.

Selecting the grammar mode

The grammar mode allows you to use only a restricted set of expressions, in either XPath or ESQL, and checks whether the syntax you have entered is valid.

WebSphere Message Broker supports the following field categories:

- Read-only path field
- Read-write path field
- Expression field

Each of these field types can be either fixed or mixed language, that is, ESQL, XPath, or either.

If you use XPath syntax, and the expressions are not supported for the property you are using, the syntax is rejected during the validation process.

ESQL and XPath have similar restrictions on the syntax that is permitted for the first two of these field types. There are restrictions to the expression fields as well, but as this type of field supports general expressions that can be used in either language, the range of syntax available is greater than in the first two.

WebSphere Message Broker uses code assistance in the grammar management of XPath 1.0 to validate the syntax of expressions you enter. This assistance is always available, regardless of the grammar mode you are using.

By default, you are operating in the restricted grammar mode.

Code assistance enables you to construct syntactically correct expressions but it does not validate those expressions. Validation is performed by property editors in which such expressions are entered.

If you attempt to use an expression that is not valid, the property editor marks it as such, either from a syntax or schema validation perspective.

You receive error or warning messages depending on the preference choices you set in **Windows>Preferences>Broker Development>XPath>Validation** .

If, under the above validation settings, particular checks are to be marked as errors, error markers are shown in the problems viewer. This behavior results in a message flow being marked as broken, and it cannot then be imported into, or compiled in, a deployable broker archive (BAR) file using the Broker Archive editor.

If you want to use the appropriate unrestricted grammar to input a specific field type, property editors do not force restricted forms of ESQL or XPath 1.0 expressions for such fields that expect them. Instead, you can enter the full range of syntax in the context of the field category concerned, namely, path or general expression, without the validation checks being applied. This means that if you need to, you can deploy the full range of syntax supported by the ESQL or XPath 1.0 runtime environment. Note, however, that such expressions might not be in a form that can be converted to the other language.

To use unrestricted grammar, carry out the following procedure:

1. Switch to the Broker Administration perspective.
2. Click **Window -> Preferences** and expand **Broker Development**.
3. Expand **XPath** and click **Grammar**.
4. Clear the **Use XPath and ESQL equivalent grammar** check box.

Note that expressions are still checked for valid syntax appropriate in the context of the field type, but you can now use the full range of grammar supported by the runtime environment.

Using TCP/IP in message flows

You can use WebSphere Message Broker to connect to applications that use raw TCP/IP sockets for transferring data.

If you have existing applications that use raw TCP/IP sockets for transferring data, you can use the WebSphere Message Broker TCP/IP nodes to connect to the applications, without needing to enable them for WebSphere MQ.

WebSphere Message Broker implements access to the TCP/IP input and output streams through the following nodes:

- "TCPIPClientInput node" on page 1213
- "TCPIPClientOutput node" on page 1225

- "TCPIPClientReceive node" on page 1234
- "TCPIPServerInput node" on page 1245
- "TCPIPServerOutput node" on page 1256
- "TCPIPServerReceive node" on page 1264

For information on how to use the TCP/IP support, see "Working with TCP/IP" on page 498.

The following topics contain information that you need to understand before you can use TCP/IP in a WebSphere Message Broker application:

- "WebSphere Broker TCP/IP Transport"
- "TCPIP nodes" on page 489
- "Connection management" on page 492
- "Scenarios for WebSphere Message Broker and TCP/IP" on page 494

WebSphere Broker TCP/IP Transport

The WebSphere Broker TCP/IP Transport is a service that connects applications that use raw TCP/IP sockets for transferring data.

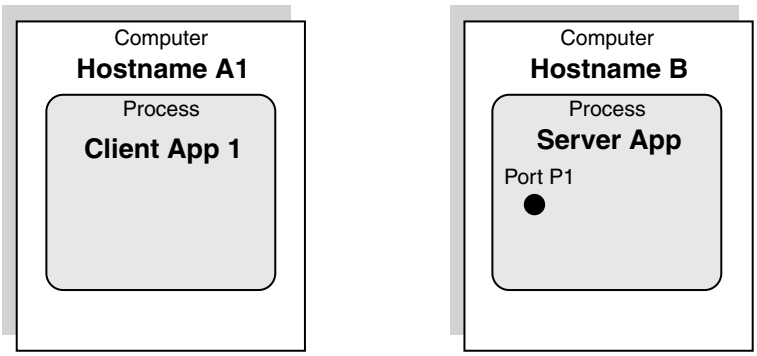
TCP/IP sockets provide a simple way of connecting computer programs together, and this type of interface is commonly added to existing stand-alone applications. TCP/IP provides a mechanism for transferring data between two applications, which can be running on different computers. The transfer of data is bidirectional and, as long as the TCP/IP connection is maintained, no data is lost, and the sequence of the data is kept. A significant advantage of using TCP/IP directly is that it is quick and simple to configure, which makes it a useful mechanism for processes that do not require message persistence (for example, monitoring).

However, the use of TCP/IP sockets for transferring information between programs does have some limitations:

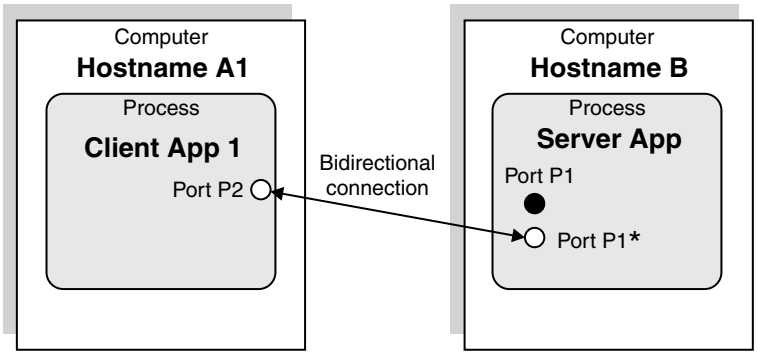
- It is non-transactional
- It is not persistent (the data is written to an in-memory buffer between the sender and receiver)
- It has no built-in security
- It provides no standard way of signaling the start and end of a message

For these reasons, it can be preferable to use a transport mechanism like WebSphere MQ, which has none of these limitations. However, if you have existing applications that use raw TCP/IP sockets for transferring data, you can use the WebSphere Message Broker TCPIP nodes to connect to the applications without needing to enable them for WebSphere MQ, so that you can develop a WebSphere Message Broker solution quickly.

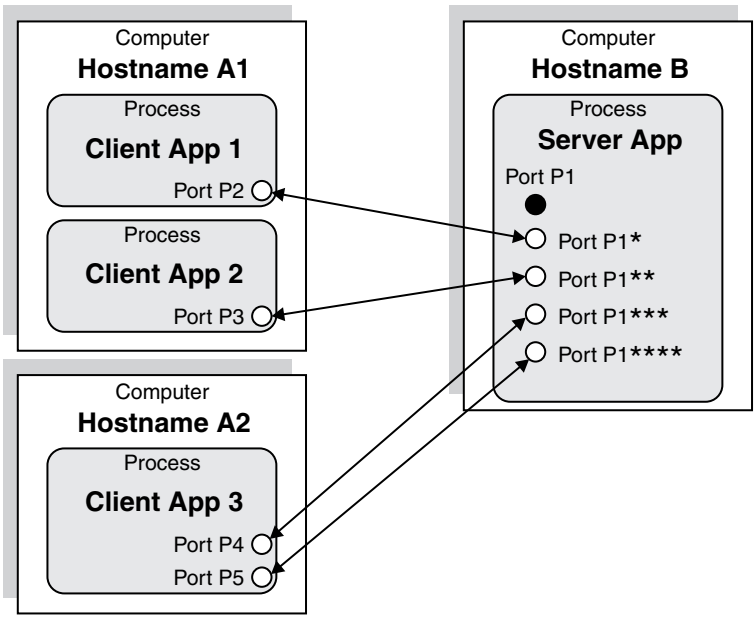
A TCP/IP connection between two applications has a client end and a server end, which means that one application acts as a server and the other as a client. The terms client and server refer only to the mechanism used to establish a connection; they do not refer to the pattern of data exchange. When the connection has been established, both client and server can perform the same operations and can both send and receive data. The following diagram illustrates the locations of client and server applications:



1. The server application listens on a local port (on the computer that is running the application) for requests for connections to be made by a client application.
2. The client application requests a connection from the server port, which the server then accepts.
3. When the server accepts the request, a port is created on the client computer and is connected to the server port.
4. A socket is created on both ends of the connection, and the details of the connection are encapsulated by the socket.
5. The server port remains available to listen for further connection requests:

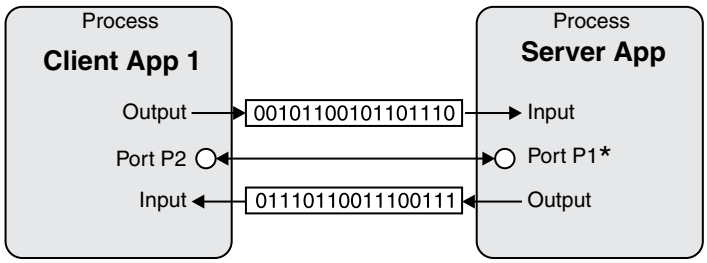


The server can accept more connections from other client applications. These connections can be in the same process, in a different process on the same computer, or on a different computer:



Only one server application can exist, but any number of different client processes can connect to the server application. Any of these applications (client or server) can be multithreaded, which enables them to use multiple connections.

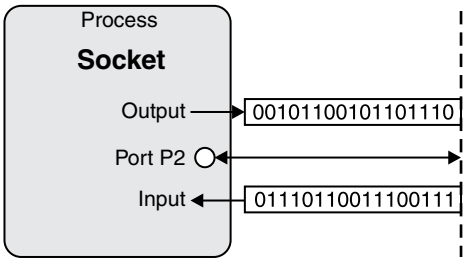
When the connection has been established, two data streams exist: one for inbound data and another for outbound data:



The client and server ends of the connection are identical and both can perform the same operations. The only difference between them is that the output stream of the client is the input stream of the server, and the input stream of the client is the output stream of the server.

The two streams of data are independent and can be accessed simultaneously from both ends. The client does not need to send data before the server.

The example illustrated in the previous diagram can be simplified in the following way, showing that the client and server have access to a socket that has an input stream and an output stream:



TCPIP nodes

WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes.

Two sets of TCPIP nodes exist: server nodes and client nodes. Both sets have identical function in terms of accessing the data streams; the only difference between them is that one set uses client connections and the other set uses server connections. As a result, the nodes establish the connections in different ways but they use the streams in the same way when the connections have been established.

The main difference between the properties of the nodes is that the server nodes do not allow the host name to be changed (because it must always be localhost). All server nodes that use the same port must be in the same execution group because the port is tied to the running process. Client nodes on the same port can be used in different execution groups, but client connections cannot be shared because the client connections are tied to a particular execution group, which maps to a process. Within the two sets of nodes (client and server) are three types of node:

- TCPIPServerInput and TCPIPClientInput
- TCPIPServerReceive and TCPIPClientReceive
- TCPIPServerOutput and TCPIPClientOutput

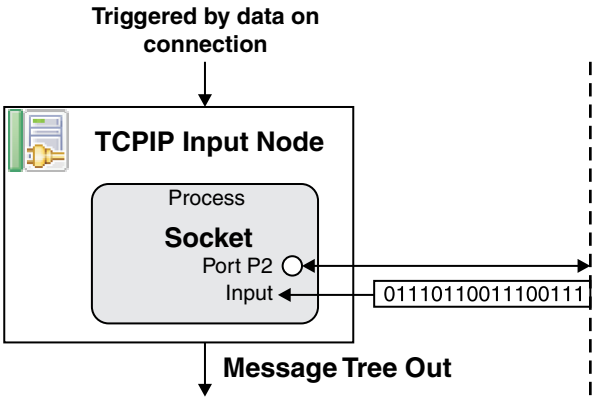
The input and receive nodes access the input stream to retrieve data, and the output nodes access the output stream to send data. No single node can access both streams at the same time. To access both streams simultaneously, you must connect multiple nodes in a message flow.

Input nodes

The input node allows access to a connection's input stream. The node is triggered by the arrival of data in the stream and starts processing the message flow. The input node controls thread and transaction management. The TCPIP nodes are not transactional in the way that they interact with TCP/IP, but other nodes in the same flow can be transactional (for example, MQ nodes). The input node does not create a thread for every connection being used but instead waits for two requirements to be met:

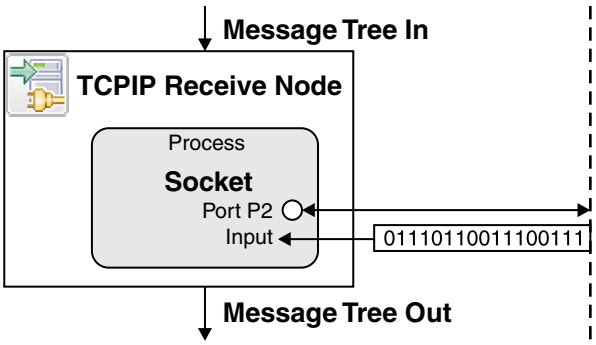
- A connection is available that still has an open input stream
- Data is available on the input stream (at least 1 byte)

For example, 1,000 TCP/IP connections can be handled by one input node that has only one additional instance. This situation is possible because the node does not poll the connections but is triggered when the specified conditions are met.



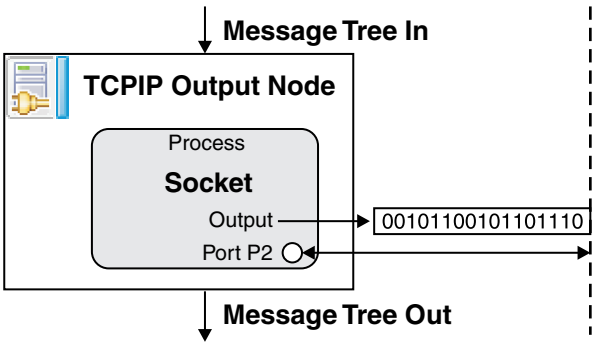
Receive nodes

The receive node is triggered to read data from a connection when a message arrives on its In terminal. It waits for data to arrive, then sends it to the Out terminal. You can configure the receive node to use a particular connection (by specifying a connection's ID) or to use any available connection. If the node is configured to use any available connection, it receives data from the first connection that has data available.



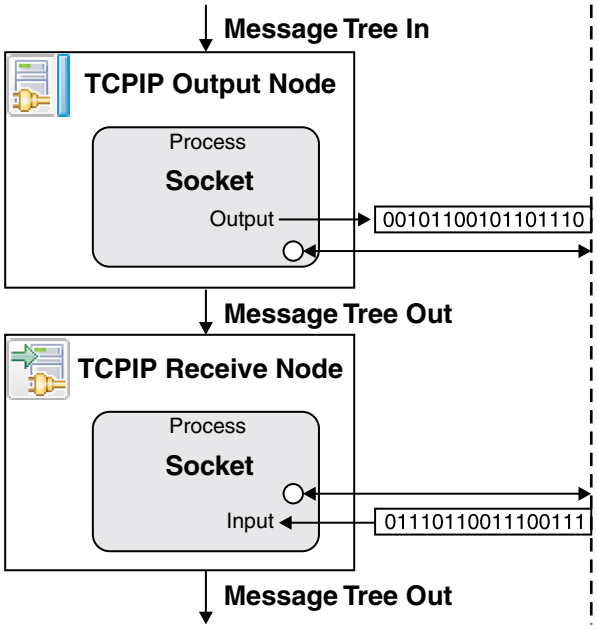
Output nodes

The output node sends data to a connection. It is triggered by a message arriving on its In terminal, then it sends the data contained in the message to the stream. The same message that is received in the node is sent out to the Out terminal.



Combining nodes

The six client and server nodes can be combined to provide more complex operations. For example, an output node followed by a receive node allows for a synchronous request of data:



If the message flows used are single threaded and only one connection ever exists, the sequence of nodes requires no further configuration. Two additional mechanisms are included to allow multithreading and multiple connections:

- A connection ID to ensure that the same connection is used by multiple nodes
- The ability to reserve connections so that they can be accessed only when the ID is specified

One connection in multiple nodes

Every connection has a unique identifier assigned to it when it is created. Whenever a node uses a connection, the ID that is used is written to the local environment. Any nodes that use it later in the flow can access the same connection by specifying the ID; the receive and output nodes find the ID by searching in a specified location in the local environment. By default, the location in which a node writes its connection details is different from the location in which the next node looks to see if there is an ID to use. The nodes can be configured to use the ID that was sent by a previous node; for example, the combination of the output and receive nodes shown in “Combining nodes” can be configured so that the receive node uses the WrittenDestination data from the preceding output node.

The use of the ID enables a series of nodes to access the same connection, but does not prevent two message flow threads accessing the same connection. When a connection is used for the first time, it can be reserved so that no other nodes can access it unless they know the ID. For example, the combination of the output and receive nodes shown in “Combining nodes” reserves the connection so that no other threads can access it before the receive node uses it. By default, the receive node then releases the connection when it has finished.

The ability to reserve connections (and access them by specifying the correct ID) enables you to build up complex interactions with TCP/IP connections that span whole flows and even multiple flows. As a result, the TCP/IP interactions can be used with other asynchronous transport mechanisms like WebSphere MQ.

Reserved connections must be released at some time, otherwise they remain unavailable indefinitely. For more information about reserved and available connections, see “Connection management.”

Correlating replies across flows

You can use the TCPIP nodes in asynchronous patterns, in which data is sent out through a TCPIP output node and received back through a TCPIP input node. The spanning of two message flows means that any state in the first flow is lost and is not accessible from the second flow. The TCPIP nodes allow you to store some reply details on a connection, and these details are then available for the input node to use when a new event arrives on the same connection. By default, this data is taken from the local environment, but you can configure the nodes to take the data from any location, including the Correlid field and message IDs in WebSphere MQ headers.

Connection management

TCP/IP connections are requested by the client connection manager and accepted by the server connection manager.

The execution group process contains the connection manager, which makes the connections. Only one execution group can have server nodes using a specific port at any one time; deployment to a second execution group causes a deployment error. Client nodes can be deployed to different execution groups, but each execution group has its own pool of connections, and therefore its own minimum and maximum number of connections.

TCP/IP nodes do not directly create or manage any TCP/IP connections, but acquire them from the connection manager's internal pool. For example, two output nodes using the same connection details share the same connection manager. The TCP/IP nodes can define the connection details to be used by specifying one of the following values:

- Host name and port
- Name of a configurable service.

If a host name and port are specified, the node uses these values when requesting connections. If a configurable service is specified, the node obtains the values for the port and host name from the values defined in the configurable service. The connection manager allows other configurable parameters in addition to the host name and port, and you can define all of these values when you are using a configurable service. When the host name and port are specified on the node, the connection manager obtains the rest of the required values from the default configurable service. However, if a configurable service is defined that is using the host name and port number, the values from that configurable service are used.

The connection manager is created when the first node that requires connections from it is deployed. The connection manager is destroyed when the last remaining node using it has been removed from the execution group (so the connection manager is no longer being used by any deployed nodes). For example, this

process can happen when existing flows are redeployed, because redeployment involves deleting all existing nodes before re-creating them.

Server connections

The server connection manager starts listening for server connections when it starts, and keeps accepting connections until the maximum number of connections (as specified in the configurable service) is reached. Any attempts to make connections after this point are refused. TCP/IP servers do not create connections; they only accept connection requests from other applications. As a result, you cannot force the creation of connections within a message flow.

Client connections

The client connection manager starts and keeps making client connections until the minimum number of connections (as defined in the configurable service) is reached. By default, the minimum number of connections is zero, which means that no connections are made. Whenever the number of connections drops below the minimum value, the connection manager starts creating more client connections. The client output and receive nodes initiate the creation of new client connections whenever none are available for them to use, unless the maximum number (as defined in the configurable service) has been reached.

Reserving and releasing connections

Each connection has an input stream and an output stream, both of which have two main states within the connection manager: available and reserved.

When a node requests a connection for input or output, without specifying the ID of a particular connection, it is given any available connection on the required stream. If no connections are available, and if the node is a client node, a new connection is made, but only if the maximum number of connections has not yet been reached. Any connection in the available state can be used by only one node at a time, but when a node has finished using it, any other node (from any flow or thread) can access it.

You can restrict access to a stream on a connection by reserving the connection. When a connection is in the reserved state, no other node can access the stream without specifying the ID of the connection. For example, an input node can request an available connection, and, when it has finished reading the data, put the stream into the reserved state. While the stream is in the reserved state, no input node (including the node that put the stream into the reserved state) can access it because input nodes can access only available streams. The only nodes that can access the stream must have the connection ID, which is written to the outgoing local environment when the data is passed down the message flow. As a result, receive nodes can read more data on the same connection, as long as the receive node is configured to use the ID from the input node's local environment.

When a connection is reserved, ownership of the connection is given to a current thread of processing. This processing can span separate message flows, if required.

The reserve mechanism provides the following options:

- Leave unchanged
- Reserve
- Release

- Reserve and release at the end of the flow

By default, the stream is left available (not reserved). This is the case for all nodes, and, for many types of processing, this default can be left unchanged; for example, when moving data from an input stream to a file. The main purpose of reserving a stream is to allow a series of nodes to be connected to give complex processing on a stream in an ordered, controlled, synchronous sequence. If you need to reserve a connection, the Reserve and release at end of flow option has the benefit of ensuring that the connection's stream is released when one iteration of the flow has finished processing, including any error conditions that might occur.

When you require the processing to span multiple message flows (for example, for asynchronous request and reply), you need to reserve a stream without releasing it at the end of the flow. See the following sample for an example:

- TCPIP Client Nodes

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

A disadvantage of reserving a stream between message flows is the potential for a stream never to be released. To avoid this problem, set an expiry time on the connection so that it is closed after a specified period of inactivity.

Another benefit of reserving an input stream is that the connection cannot be closed until it is either released or expired (even if an end application closes its end of the connection), which is useful when the end of the stream is being used to delimit messages in the stream.

File descriptors

If your WebSphere Message Broker application is running on Sun Solaris 10 on SPARC, you might need to increase the number of file descriptors. The following error in the syslog indicates that additional file descriptors are required:

```
Failed to create a client connection using hostname: '', port: ''. Reason: 'Invalid argument'
```

Scenarios for WebSphere Message Broker and TCP/IP

Two example scenarios show how you might use TCP/IP and WebSphere Message Broker as part of a business solution.

- "Scenarios: TCP/IP"
- "Scenarios: Message Broker using TCP/IP" on page 496

Scenarios: TCP/IP

Two scenarios illustrate how TCP/IP might be used as part of a business solution.

- "Expense submission"
- "Price-change notification" on page 496.

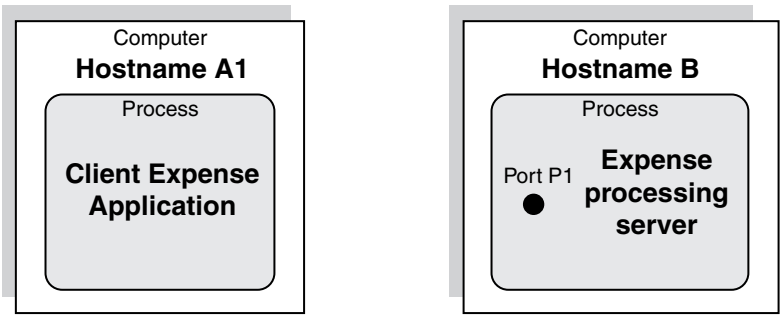
Expense submission: Scenario:

A company has an expense-submission system based on a central expense-processing application, which receives completed expense forms from end users. The users complete the forms by using a local application, which stores the form until it is completed. When it has been completed, the form is transferred to the central system where it is processed. Any further notifications are sent to the user by e-mail.

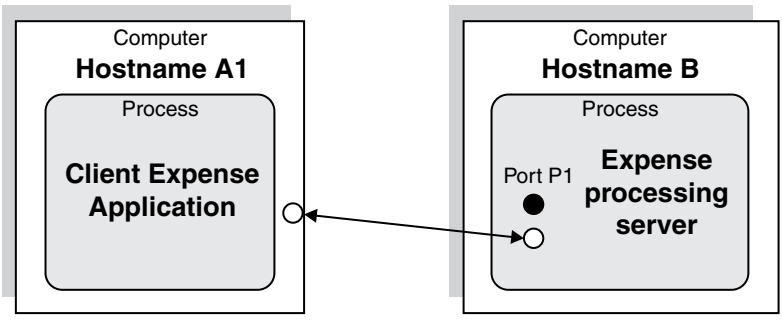
How TCP/IP is used:

One client application exists for each end user, and one central application processes all the expense forms. Each client application connects to a TCP/IP server port on the server application and sends the expense form in a fixed-field-size structure similar to a COBOL structure. The flow of processing is:

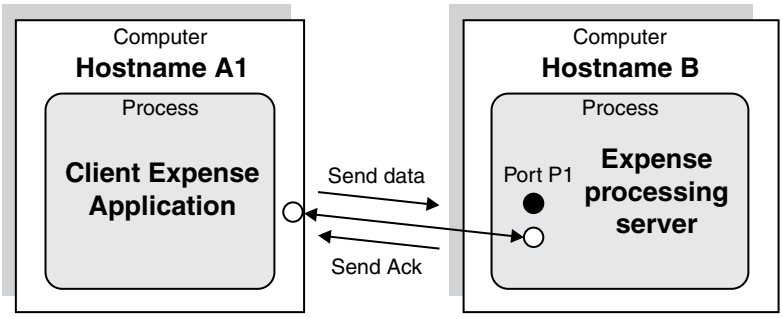
1. The user enters information into the form in the client expense application.



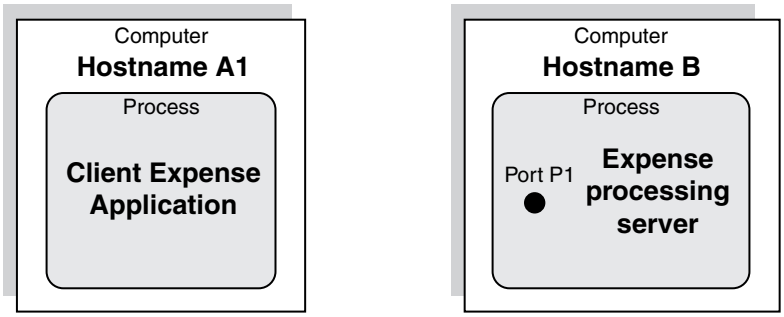
2. The user submits the completed form and the client application connects to the server.



3. The client application sends the data to the server and receives an acknowledgment.



4. The connection is closed by the client.

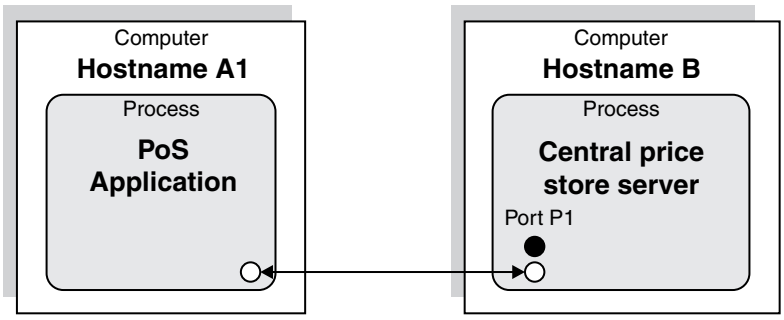


Price-change notification: Scenario:

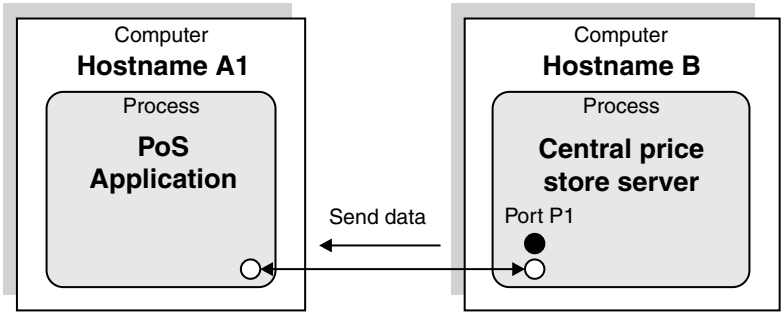
A company has a central server, which stores the catalogue price of everything that the company sells. This information is required by all Point of Sale (PoS) terminals in all the stores, and each PoS terminal must be notified when any price changes. The PoS terminals connect to the central server and wait for any process changes. The server sends any process changes to all connected client applications.

How TCP/IP is used:

1. When the PoS application starts, it connects to the central server.



2. Whenever the server has a new price, it publishes it to all the connected clients.



3. The PoS stays connected until it shuts down.

See “Scenarios: Message Broker using TCP/IP” for an example of how these scenarios can be modified to use WebSphere Message Broker.

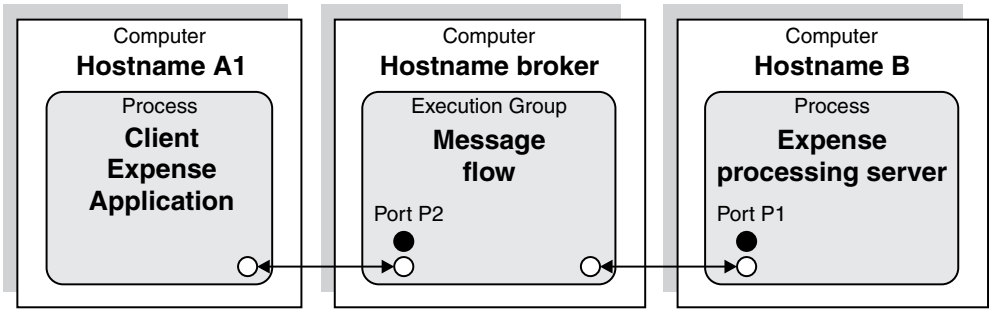
Scenarios: Message Broker using TCP/IP

WebSphere Message Broker can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

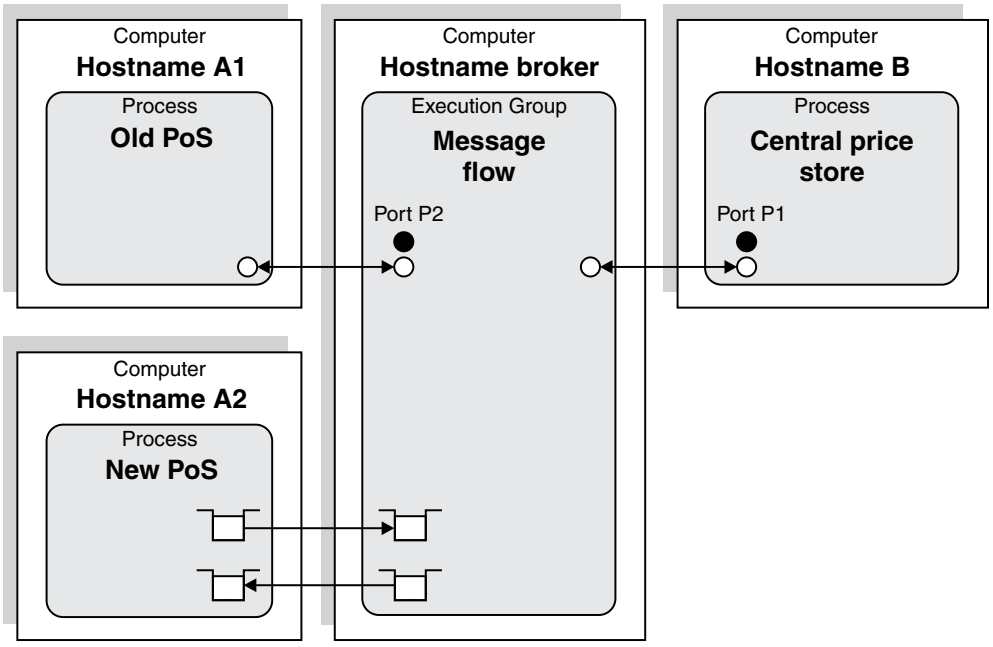
The scenarios in the "Scenarios: TCP/IP" on page 494 topic show how systems can be created to use TCP/IP as a transport mechanism. The following sections show how WebSphere Message Broker can be added to those systems to generate a more flexible architecture for communication between components:

- "Expense submission" illustrates TCP/IP to TCP/IP routing
- "Price-change notification" illustrates routing and transformation to other formats

Expense submission: The expense submission scenario shown in the "Scenarios: TCP/IP" on page 494 topic requires a direct connection from the client applications to the end server. With that model it is difficult to add new consumers of the expense submission information, and it is also difficult to change the end application that processes them. However, by adding WebSphere Message Broker as an intermediary router, the two systems can be separated without any changes to their interfaces, as shown in the following diagram:



Price-change notification: The price-change notification scenario shown in the "Scenarios: TCP/IP" on page 494 topic can be modified to use a message broker for routing and transformation. It could also allow support for other protocols like WebSphere MQ, which would allow new applications to be written to different interfaces without the need for changing the current client or server applications:



Working with TCP/IP

You can use WebSphere Message Broker TCPIP nodes and TCP/IP configurable services to perform a variety of tasks.

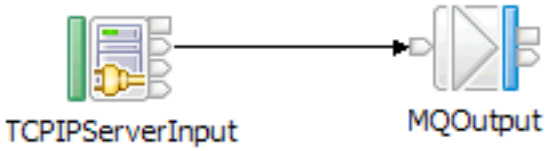
Transferring XML data from a TCP/IP server socket to a WebSphere MQ queue

Transfer XML data from a TCP/IP server socket to a WebSphere MQ queue, by creating a message flow with TCPIPServerInput and MQOutput nodes.

Scenario: A client application opens a TCP/IP socket and sends an XML document. The end of the document is signalled by the closure of the client connection.

Instructions: The following steps describe how to write a message flow that can receive the XML document and write it to a WebSphere MQ queue:

1. Create a message flow called TCPIP_Task1 with a TCPIPServerInput node and an MQOutput node. For more information about how to do this, see "Creating a message flow" on page 259.
2. Connect the Out terminal of the TCPIPServerInput node to the In terminal of the MQOutput node.



3. Set the following properties of the TCPIPServerInput node:
   a. On the **Basic** tab, set the Connection details property to 14141.
   b. On the **Input Message Parsing** tab, set the Message domain property to XMLNSC.
4. On the MQOutput node, set the Queue name property (on the **Basic** tab) to TCPIP.TASK1.OUT1.
5. Save the message flow.

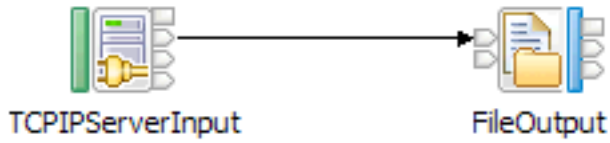
Transferring binary (CWF) data from a TCP/IP server socket to a flat file

Transfer binary Custom Wire Format (CWF) data from a TCP/IP server socket to a flat file, by the use of a message set and a message flow with TCPIPServerInput and FileOutput nodes.

Scenario: A client application opens a TCP/IP socket and sends a binary (CWF) document. The end of the document is signaled by the closure of the client connection.

Instructions: The following steps describe how to write a message flow that can receive the binary document and write it to a flat file. Each message is written to a separate file, the name of which is based on the ID of the connection.

1. Create a message set called Task2_MsgSet. For more information, see Creating a message set.
2. Create a message flow called TCPIP_Task2 with a TCPIPServerInput node and a FileOutput node. For more information, see “Creating a message flow” on page 259.
3. Connect the Out terminal of the TCPIPServerInput node to the In terminal of the FileOutput node.



4. Set the following properties of the TCPIPServerInput node:
   a. On the **Basic** tab, set the Connection details property to 14142.
   b. On the **Input Message Parsing** tab, set the following properties:
      - Set the Message domain property to MRM.
      - Set the Message set property to Task2_MsgSet.
      - Set the Message type property to Task2_MsgType.
      - Set the Message format property to Binary1.
5. Set the following properties of the FileOutput node:

a. On the **Basic** tab, set the following properties:
   - Set the Directory property to c:\temp\task2.
   - Set the File name or pattern property to Task2.out.

b. On the **Request** tab, set the Request file name property location property to $LocalEnvironment/TCPIP/Input/ConnectionDetails/Id.

6. Save the message flow.
7. Create a project reference between the message flow project and the message set project:
   a. Right-click the message flow project, and click **Properties**.
   b. Click **Project References**.
   c. Select the message set project that contains your message set (Task2_MsgSet).
   d. Click **OK**.

Receiving data on a TCP/IP server socket and sending data back to the same connection

Receive data on a TCP/IP server socket, then send the data to the same connection, by the use of a message flow with TCPIPServerInput and TCPIPServerOutput nodes.

Scenario: A client application opens a TCP/IP socket and sends an undefined document (of any format or size). The end of the document is signalled by the client closing the output stream (but not the connection), and waiting for the same data to be sent back.

Instructions: The following steps describe how to write a message flow that can receive the data and echo it back to the same connection:

1. Create a message flow called TCPIP_Task3 with a TCPIPServerInput node and a TCPIPServerOutput node. For more information, see "Creating a message flow" on page 259.
2. Connect the Out terminal of the TCPIPServerInput node to the In terminal of the TCPIPServerOutput node.

3. Set the following properties of the TCPIPServerInput node:
   a. On the **Basic** tab, set the Connection details property to 14143.
   b. On the **Advanced** tab, set the Input stream modification property to Reserve input stream and release at end of flow.
4. Set the following properties of the TCPIPServerOutput node:
   a. On the **Basic** tab, set the Connection details property to 14143.
   b. On the **Request** tab, set the ID location property to LocalEnvironment/TCPIP/Input/ConnectionDetails/Id.
   c. On the **Advanced** tab, set the Close connection property to After data has been sent.
5. Save the message flow.

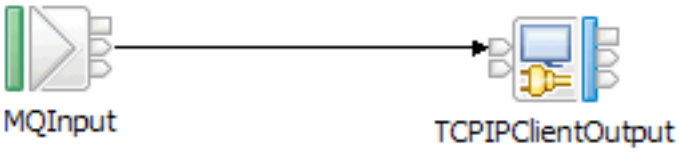
Sending XML data from a WebSphere MQ queue to a TCP/IP client socket

Send XML data from a WebSphere MQ queue to a TCP/IP client socket, by the use of a message flow with MQInput and TCPIPClientOutput nodes.

Scenario: A server application listens on a TCP/IP socket and waits for a TCP/IP client to connect and send data. The end of the document is signalled by the client closing the connection.

Instructions: The following steps describe how to write a message flow that can take a message from a WebSphere MQ queue, make the client connection, and send the data to the server application:

1. Create a message flow called TCPIP_Task4 with an MQInput node and a TCPIPClientOutput node. For more information, see "Creating a message flow" on page 259.
2. Connect the Out terminal of the MQInput node to the In terminal of the TCPIPClientOutput node.



3. Set the following properties of the MQInput node:
   a. On the **Basic** tab, set the Queue name property to TCPIP.TASK4.IN1.
   b. On the **Input message parsing** tab, set the Message domain property to XMLNSC.
4. Set the following properties of the TCPIPClientOutput node:
   a. On the **Basic** tab, set the Connection details property to 14144.
   b. On the **Advanced** tab, set the Close connection property to After data has been sent.
5. Save the message flow.

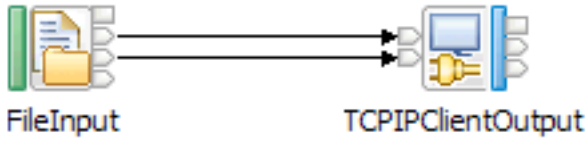
Sending CWF data from a flat file to a TCP/IP client socket

Send Custom Wire Format (CWF) data from a flat file to a TCP/IP client socket, by the use of a message flow with FileInput and TCPIPClientOutput nodes.

Scenario: An application writes 100–byte binary records into a flat file.

Instructions: The following steps describe how to open a new client TCP/IP connection and send the binary data with a binary termination character x'00FF'. When the whole file is finished, the client connection is closed:

1. Create a message flow called TCPIP_Task5 with a FileInput node and a TCPIPClientOutput node. For more information, see "Creating a message flow" on page 259.
2. Connect the Out terminal of the FileInput node to the In terminal of the TCPIPClientOutput node.
3. Connect the End of Data terminal of the FileInput node to the Close terminal of the TCPIPClientOutput node.



4. Set the following properties of the FileInput node:
   a. On the **Basic** tab, set the Input directory property to c:\temp\task5.
   b. On the **Records and elements** tab, set the following properties:
      - Set the Record detection property to Fixed length.
      - Set the Length property to 100.
5. Set the following properties of the TCPIPClientOutput node:
   a. On the **Basic** tab, set the Connection details property to 14145.
   b. On the **Advanced** tab, set the Close connection property to After data has been sent.
   c. On the **Records and elements** tab, set the following properties:
      - Set the Record definition property to Record is delimited data.
      - Set the Delimiter property to Custom delimiter (Hexadecimal).
      - Set the Custom delimiter property to 00FF.
6. Save the message flow.

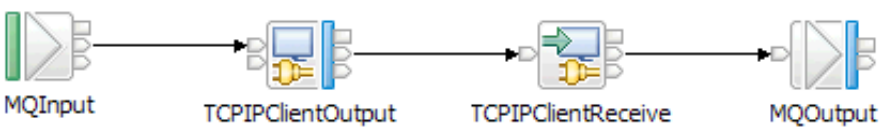
Sending data to a TCP/IP client connection and receiving data back on the same connection (synchronous)

Send fixed-size data to a TCP/IP client connection and receive fixed-size data back on the same connection (synchronously), by the use of a message flow with MQInput, TCPIPClientOutput, TCPIPClientReceive, and MQOutput nodes.

Scenario: An application sends synchronous data between TCP/IP client connections.

Instructions: The following steps describe how to create a message flow that sends data out from a client connection and waits on the same connection for a reply to be returned. The request is synchronous in the same flow, because the TCPIPClientReceive node waits for data to be returned.

1. Create a message flow called TCPIP_Task6 with an MQInput node, a TCPIPClientOutput node, a TCPIPClientReceive node, and an MQOutput node. For more information, see "Creating a message flow" on page 259.
2. Connect the Out terminal of the MQInput node to the In terminal of the TCPIPClientOutput node.
3. Connect the Out terminal of the TCPIPClientOutput node to the In terminal of the TCPIPClientReceive node.
4. Connect the Out terminal of the TCPIPClientReceive node to the In terminal of the MQOutput node.



5. On the MQInput node, set the Queue name property (on the **Basic** tab) to TCPIP.TASK6.IN1.
6. Set the following properties of the TCPIPClientOutput node:
   a. On the **Basic** tab, set the Connection details property to 14146.
   b. On the **Records and elements** tab, set the following properties:
      - Set the Record detection property to Fixed length.
      - Set the Length property to 100.
7. Set the following properties of the TCPIPClientReceive node:
   a. On the **Basic** tab, set the Connection details property to 14146.
   b. On the **Advanced** tab, set the following properties:
      - Set the Output stream modification property to Reserve output stream and release at end of flow.
      - Set the Input stream modification property to Reserve input stream and release at end of flow.
   c. On the **Request** tab, set the ID location property to $LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails[1]/Id.
   d. On the **Records and elements** tab, set the following properties:
      - Set the Record detection property to Fixed length.
      - Set the Length property to 100.
8. On the MQOutput node, set the Queue name property (on the **Basic** tab) to TCPIP.TASK6.OUT1.
9. Save the message flow.

See the following sample for more information:

- TCPIP Client Nodes

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

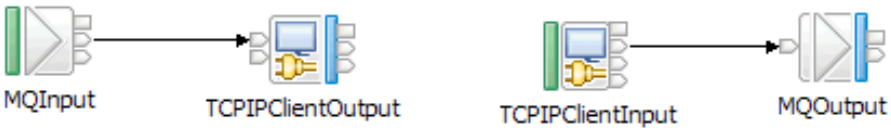
Sending data to a TCP/IP client connection and receiving data back on the same connection (asynchronous)

Send fixed-size data to a TCP/IP client connection and receive fixed-size data back on the same connection (asynchronously), by the use of a message flow with MQInput, TCPIPClientOutput, TCPIPClientInput, and MQOutput nodes.

Scenario: An application sends asynchronous data between TCP/IP client connections.

Instructions: The following steps describe how to create a message flow to send data through a client connection and wait on the same connection for a reply to be returned. The request is performed asynchronously in two different flows (the TCPIPClientInput does not wait for data to be returned on this connection, but instead monitors all available connections).

1. Create a message flow called TCPIP_Task7 with an MQInput node, a TCPIPClientOutput node, a TCPIPClientInput node, and an MQOutput node. For more information, see “Creating a message flow” on page 259.
2. Connect the Out terminal of the MQInput node to the In terminal of the TCPIPClientOutput node.
3. Connect the Out terminal of the TCPIPClientInput node to the In terminal of the MQOutput node.



4. On the MQInput node, set the Queue name property (on the **Basic** tab) to TCPIP.TASK7.IN1.
5. Set the following properties of the TCPIPClientOutput node:
   a. On the **Basic** tab, set the Connection details property to 14147.
   b. On the **Advanced** tab, set the Output stream modification property to Reserve output stream.
   c. On the **Records and elements** tab, set the following properties:
      - Set the Record definition property to Fixed length.
      - Set the Length property to 100.
6. Set the following properties of the TCPIPClientInput node:
   a. On the **Basic** tab, set the Connection details property to 14147.
   b. On the **Advanced** tab, set the Output stream modification property to Release output stream and reset Reply ID.
   c. On the **Records and elements** tab, set the following properties:
      - Set the Record detection property to Fixed length.
      - Set the Length property to 100.
7. On the MQOutput node, set the Queue name property (on the **Basic** tab) to TCPIP.TASK7.OUT1.
8. Save the message flow.

See the following sample for more information:

- TCPIP Client Nodes

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

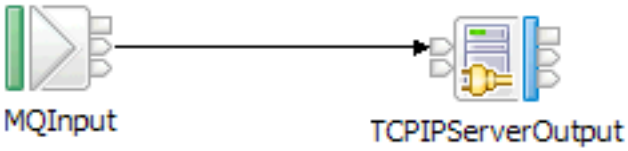
Broadcasting data to all currently available connections

Broadcast data to all current connections, by the use of a message flow with MQInput and TCPIPServerOutput nodes.

Scenario: Several applications connect into the message flow and wait to be sent data.

Instructions: The following steps describe how to send data to all the connected client applications:

1. Create a message flow called TCPIP_Task8 with an MQInput node and a TCPIPServerOutput node. For more information, see “Creating a message flow” on page 259.
2. Connect the Out terminal of the MQInput node to the In terminal of the TCPIPServerOutput node.



3. On the MQInput node, set the Queue name property (on the **Basic** tab) to TCPIP.TASK8.IN1.
4. Set the following properties of the TCPIPServerOutput node:
   a. On the **Basic** tab, set the Connection details property to 14148.
   b. On the **Advanced** tab, set the Send to: property (in the **Broadcast options** group) to All available connections.
5. Save the message flow.

Configuring a server socket so that connections expire after a specified time

Configure a TCP/IP server socket so that connections expire after a specified time, by the use of the mqsicreateconfigurableservice command.

Scenario: The TCPIPServer configurable service called Task9 is configured to run on port 14149. The connections expire when they have not been used for 5 seconds.

Instructions: Use the mqsicreateconfigurableservice command to set up connections that expire when they have not been used for a specified length of time. The TCP/IP node can specify either the port to be used or the name of the configurable service. For example:

```
mqsicreateconfigurableservice BRK6 -c TCPIPServer -o Task9
-n Port,ExpireConnectionSec -v 14149,5
```

Configuring a client socket to make 100 connections at deployment or startup time

Configure a TCP/IP client socket to make 100 connections at deployment or startup time, by using the mqsicreateconfigurableservice command.

Use the mqsicreateconfigurableservice command to set up the client Configuration Manager to establish 100 connections when it is created. By default, the client connections are not made until they are required by one of the TCP/IP nodes. For example:

```
mqsicreateconfigurableservice WBRK_BROKER -c TCPIPClient -o Task10
-n Port,MinimumConnections -v 14150,100
```

In this example, the TCPIPClient configurable service called Task10 is configured to run on port 14150, and 100 connections are created.

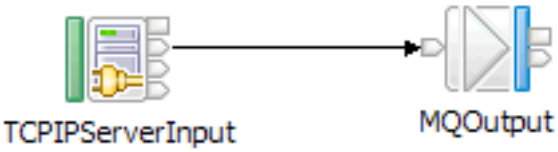
Configuring a server socket to receive XML data ending in a null character

Configure a server TCP/IP socket to receive XML data ending in a null character, by the use of a message flow with TCPIPServerInput and MQOutput nodes.

Scenario: A client application sends XML data that is delimited by a null character (hex code '00').

Instructions: The following steps describe how to break up the record based on the null character, then parse the data.

1. Create a message flow called TCPIP_Task11 with a TCPIPServerInput node and an MQOutput node. For more information, see “Creating a message flow” on page 259.
2. Connect the Out terminal of the TCPIPServerInput node to the In terminal of the MQOutput node.



3. Set the following properties of the TCPIPServerInput node:
   a. On the **Basic** tab, set the Connection details property to 14151.
   b. On the **Input Message Parsing** tab, set the Message domain property to XMLNSC.
   c. On the **Records and elements** tab, set the following properties:
      - Set the Record detection property to Delimited.
      - Set the Delimiter property to Custom delimiter.
      - Set the Custom delimiter property to 00.
4. On the MQOutput node, set the Queue name property (on the **Basic** tab) to TCPIP.TASK11.IN1.
5. Save the message flow.

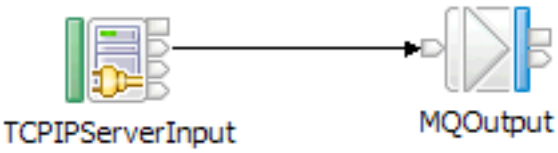
Configuring a server socket to receive XML data and discover the end of a record (by using the message model)

Configure a server socket to receive XML data and use the message model to determine the end of a record, by the use of a message flow with TCPIPServerInput and MQOutput nodes.

Scenario: A client application sends an XML document with no clear indication of the end of the record.

Instructions: The following steps show how to break up the record by the use of the XML parser to signal when the whole XML document as been received. The parser uses the end XML tag to signal the end of the message.

1. Create a message flow called TCPIP_Task12 with a TCPIPServerInput node and an MQOutput node. For more information, see “Creating a message flow” on page 259.
2. Connect the Out terminal of the TCPIPServerInput node to the In terminal of the MQOutput node.



3. Set the following properties of the TCPIPServerInput node:
   a. On the **Basic** tab, set the Connection details property to 14151.

b. On the **Input Message Parsing** tab, set the Message domain property to XMLNSC.

c. On the **Records and elements** tab, set the Record detection property to Parsed record sequence.

4. On the MQOutput node, set the Queue name property (on the **Basic** tab) to TCPIP.TASK12.IN1.
5. Save the message flow.

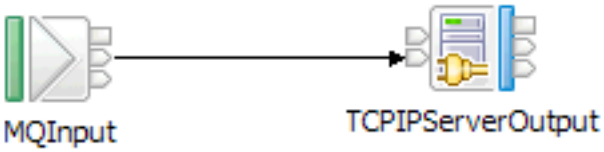
Configuring a server output node to close all connections

Configure a server output node to close all connections on a specified port.

Scenario: A server output node is configured to close all connections on a specified port.

Instructions: The following steps show how to create a message flow that closes all TCP/IP connections on port 14153:

1. Create a message flow called TCPIP_Task13 with an MQInput node and a TCPIPServerOutput node. For more information, see "Creating a message flow" on page 259.
2. Connect the Out terminal of the MQInput node to the Close terminal of the TCPIPServerOutput node.



3. On the MQInput node, set the Queue name property (on the **Basic** tab) to TCPIP.TASK13.IN1.
4. Set the following properties of the TCPIPServerOutput node:
   a. On the **Basic** tab, set the Connection details property to 14153.
   b. On the **Advanced** tab, set the Send to: property (in the **Broadcast options** group) to All available connections.
5. Save the message flow.

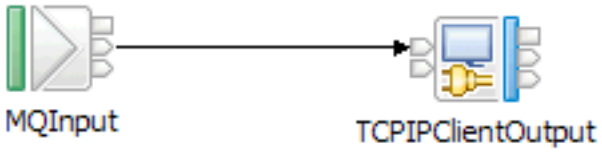
Configuring a client socket to store reply correlation details

Configure a client TCP/IP socket to store reply correlation details, by the use of a message flow with MQInput and TCPIPClientOutput nodes.

Scenario: A client TCP/IP socket is configured to store response returned on the input stream.

Instructions: The following steps show how to set the Reply ID on a connection, which can be used when a response is returned on the input stream:

1. Create a message flow called TCPIP_Task14 with an MQInput node and a TCPIPClientOutput node. For more information, see "Creating a message flow" on page 259.
2. Connect the Out terminal of the MQInput node to the In terminal of the TCPIPClientOutput node.



3. On the MQInput node, set the Queue name property (on the **Basic** tab) to TCPIP.TASK14.IN1.
4. Set the following properties of the TCPIPClientOutput node:
   a. On the **Basic** tab, set the Connection details property to 14154.
   b. On the **Request** tab, set the Reply ID location property to $Root/MQMD/MsgId.
5. Save the message flow.

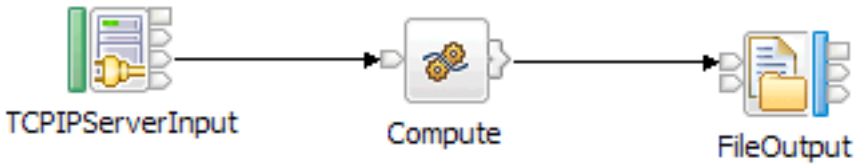
Writing close connection details to a file

Configure a message flow to write details of a connection closure to a file, by the use of TCPIPServerInput, Compute, and FileOutput nodes.

Scenario: A message flow writes close connection details to a file.

Instructions The following steps show how to configure a message flow to write details of the closure of any connection to a file:

1. Create a message flow called TCPIP_Task15 with a TCPIPServerInput node, a Compute node, and a FileOutput node. For more information, see "Creating a message flow" on page 259.
2. Connect the Close terminal of the TCPIPServerInput node to the In terminal of the Compute node.
3. Connect the Out terminal of the Compute node to the In terminal of the FileOutput node.



4. On the TCPIPServerInput node, set the Connection details property (on the **Basic** tab) to 14155.
5. On the Compute node, set the ESQL property (on the **Basic** tab) to:

```
BROKER SCHEMA Tasks


CREATE COMPUTE MODULE TCPIP_Task15_Compute
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  -- CALL CopyMessageHeaders();
  -- CALL CopyEntireMessage();
  Set OutputRoot.XMLNSC.CloseEvent = InputLocalEnvironment.TCPIP;
  RETURN TRUE;
 END;

 CREATE PROCEDURE CopyMessageHeaders() BEGIN
  DECLARE I INTEGER 1;
```

```
 DECLARE J INTEGER;
 SET J = CARDINALITY(InputRoot.*[]);
 WHILE I < J DO
  SET OutputRoot.*[I] = InputRoot.*[I];
  SET I = I + 1;
 END WHILE;
END;

CREATE PROCEDURE CopyEntireMessage() BEGIN
 SET OutputRoot = InputRoot;
END;
END MODULE;
```

6. Set the following properties of the FileOutput node.
   a. On the **Basic** tab, set the following properties:
      - Set the Directory property to c:\temp\Task15.
      - Set the File name or pattern property to CloseEvents.txt.
   b. On the **Records and elements** tab, set the Record definition property to Record is unmodified data.
7. Save the message flow.

Configuring a client node to dynamically call a port

Configure a client node to dynamically use a port and hostname that are set in the local environment, by the use of a message flow with MQInput, Compute, and TCPIPClientOutput nodes.

Scenario: A client node dynamically calls a port.

Instructions: The following steps show how to override the connection details specified on a client output node to dynamically use a port and hostname that are set in the local environment:

1. Create a message flow called TCPIP_Task16 with an MQInput node, a Compute node, and a TCPIPClientOutput node. For more information, see "Creating a message flow" on page 259.
2. Connect the Out terminal of the MQInput node to the In terminal of the Compute node.
3. Connect the Out terminal of the Compute node to the In terminal of the TCPIPClientOutput node.

4. On the MQInput node, set the Queue name property (on the **Basic** tab) to TCPIP.TASK16.IN1 .
5. On the Compute node, set the ESQL property (on the **Basic** tab) to:

```
BROKER SCHEMA Tasks
CREATE COMPUTE MODULE TCPIP_Task16_Compute
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  -- CALL CopyMessageHeaders();
  CALL CopyEntireMessage();
  set InputLocalEnvironment.Destination.TCPIP.Output.Hostname = 'localhost';
  set InputLocalEnvironment.Destination.TCPIP.Output.Port = 14156;
```

```
 RETURN TRUE;
END;

CREATE PROCEDURE CopyMessageHeaders() BEGIN
 DECLARE I INTEGER 1;
 DECLARE J INTEGER;
 SET J = CARDINALITY(InputRoot.*[]);
 WHILE I < J DO
  SET OutputRoot.*[I] = InputRoot.*[I];
  SET I = I + 1;
 END WHILE;
END;

CREATE PROCEDURE CopyEntireMessage() BEGIN
 SET OutputRoot = InputRoot;
END;
END MODULE;
```

6. On the TCPIPClientOutput node, set the Connection details property (on the **Basic** tab) to 9999.
7. Save the message flow.

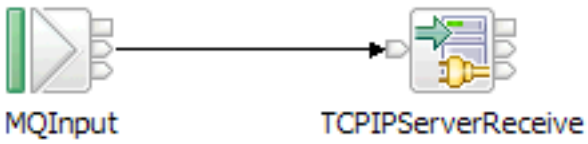
Configuring a server receive node to wait for data on a specified port

Configure a TCPIPServerReceive node to block the message flow and wait for data to arrive on any connection.

Scenario: A server receive node is configured to wait for data on a specified port.

Instructions: The following steps show how to configure a TCPIPServerReceive node to wait for data on port 14157:

1. Create a message flow called TCPIP_Task17 with an MQInput node and a TCPIPServerReceive node. For more information, see "Creating a message flow" on page 259.
2. Connect the Out terminal of the MQInput node to the In terminal of the TCPIPServerReceive node.



3. On the MQInput node, set the Queue name property (on the **Basic** tab) to TCPIP.TASK17.IN1.
4. On the TCPIPServerReceive node, set the Connection details property (on the **Basic** tab) to 14157.
5. Save the message flow.

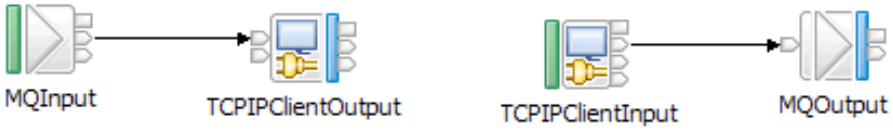
Sending and receiving data through a TCP/IP client connection, delimiting the record by closing the output stream (asynchronous)

Send data through a TCP/IP client connection and receive data back on the same connection (asynchronously), by the use of a message flow with MQInput, TCPIPClientOutput, TCPIPClientInput, and MQOutput nodes.

Scenario: An application sends asynchronous data between TCP/IP client connections.

Instructions: The following steps describe how to create a message flow to send data through a client connection and wait on the same connection for a reply to be returned. The request is performed asynchronously in two different flows; the TCPIPClientInput node does not wait for data to be returned on this connection, but monitors all available connections. The outgoing record is delimited by closing the output stream, and the reply message is delimited by the remote server closing the input stream. The connection is then completely closed by the node.

1. Create a message flow called TCPIP_Task18 with an MQInput node, a TCPIPClientOutput, a TCPIPClientInput node, and an MQOutput node. For more information, see "Creating a message flow" on page 259.
2. Connect the Out terminal of the MQInput node to the In terminal of the TCPIPClientOutput node.
3. Connect the Out terminal of the TCPIPClientInput node to the In terminal of the MQOutput node.



4. On the MQInput node, set the Queue name property (on the **Basic** tab) to TCPIP.TASK18.IN1.
5. Set the following properties of the TCPIPClientOutput node:
   a. On the **Basic** tab, set the Connection details property to 14158.
   b. On the **Advanced** tab, select Close output stream after a record has been sent.
   c. On the **Records and elements** tab, set the Record definition property to Record is unmodified data.
6. Set the following properties of the TCPIPClientInput node:
   a. On the **Basic** tab, set the Connection details property to 14158.
   b. On the **Advanced** tab, set the Close connection property to After data has been received.
   c. On the **Records and elements** tab, set the Record detection property to End of stream.
7. On the MQOutput node, set the Queue name property (on the **Basic** tab) to TCPIP.TASK18.OUT1.
8. Save the message flow.

Sending and receiving data on the same TCP/IP client connection, closing input and output streams (synchronous)

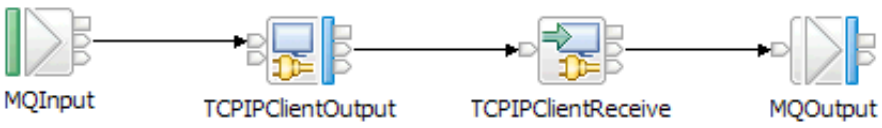
Send data through a TCP/IP client connection and wait on the same connection for a reply to be returned, by the use of a message flow with MQInput, TCPIPClientOutput, TCPIPClientReceive, and MQOutput nodes.

Scenario: An application sends synchronous data on the same TCP/IP client connection.

Instructions: The following steps describe how to create a message flow that sends out data through a client connection and waits on the same connection for a reply

to be returned. The request is synchronous within the same flow, as a result of the TCPIPClientReceive node waiting for data to be returned. The outgoing message is delimited by closing the output stream, and the reply data is delimited by the remote application closing the input stream.

1. Create a message flow called TCPIP_Task19 with an MQInput node, a TCPIPClientOutput node, a TCPIPClientReceive node, and an MQOutput node. For more information, see "Creating a message flow" on page 259.
2. Connect the Out terminal of the MQInput node to the In terminal of the TCPIPClientOutput node.
3. Connect the Out terminal of the TCPIPClientOutput node to the In terminal of the TCPIPClientReceive node.
4. Connect the Out terminal of the TCPIPClientReceive node to the In terminal of the MQOutput node.



5. On the MQInput node, set the Queue name property (on the **Basic** tab) to TCPIP.TASK19.IN1.
6. Set the following properties of the TCPIPClientOutput node:
   a. On the **Basic** tab, set the Connection details property to 14159.
   b. On the **Advanced** tab, set the following properties:
      - Select Close output stream after a record has been sent.
      - Set the Input stream modification property to Reserve input stream and release at end of flow. It is important to reserve the input stream so that it is not closed before the receive node processes the return data.
   c. On the **Records and elements** tab, set the Record definition property to Record is Unmodified Data.
7. Set the following properties of the TCPIPClientReceive node:
   a. On the **Basic** tab, set the Connection details property to 14159.
   b. On the **Advanced** tab, set the Close connection property to After data has been received.
   c. On the **Request** tab, set the ID location property to $LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails[1]/Id.
   d. On the **Records and elements** tab, set the Record detection property to Connection closed.
8. On the MQOutput node, set the Queue name property (on the **Basic** tab) to TCPIP.TASK19.OUT1.
9. Save the message flow.

Sending e-mails

You can configure the EmailOutput node to send an e-mail, with or without attachments, to a static or dynamic list of recipients.

You can use the SMTP Server and Port property of the EmailOutput node to specify the host name of the SMTP server that the broker uses to send e-mails, or to specify an alias that refers to a configurable service that is defined on the broker.

To enable the configurable service, use the mqsicreateconfigurableservice and mqsichangeproperties commands, as shown in the following example:

```
mqsicreateconfigurableservice BROKER -c SMTP -o MYSERVER
mqsichangeproperties BROKER -c SMTP -o MYSERVER -n serverName -v smtp.hursley.ibm.com:25
```

If the value of the SMTP Server and Port property is set to the defined alias (in this case, MYSERVER), any values that are overridden by the administrator (smtp.hursley.ibm.com:25 set on the command line as the server name) are used in preference to any statically defined value or override value in the local environment.

The order of preference for value selection is:

1. The value that is specified in the configurable service if an alias exists with the name that was supplied in the SMTP Server and Port property.
2. The server name that is specified in the local environment.
3. The value of the SMTP Server and Port property that is specified on the node.

Any alias can be removed by using the mqsideleteconfigurableservice command, so that the node reverts to resolving the host name from the value that is set in the local environment or on the node. While most configurable service properties are set by the mqsichangeproperties command, security identities (user IDs and passwords) are typically set by using the mqsisetdbparms command and are referenced in a separate SecurityIdentity property, for example:

```
mqsisetdbparms -n smtp::MyIdentity -u userName -p password
mqsichangeproperties BROKER -c SMTP -o MYSERVER -n securityIdentity -v MyIdentity
```

The topics in this section describe the different ways in which you can use the EmailOutput node to send e-mail messages.

- To send an e-mail with a static subject and static text to a static list of recipients, see "Sending an e-mail."
- To send an e-mail with an attachment, see "Sending an e-mail with an attachment" on page 514.
- To send an e-mail where the SMTP server, list of recipients, subject, text, and multiple attachments are all determined at run time, see "Producing dynamic e-mail messages" on page 515.
- To send an e-mail that is constructed from a MIME message, see "Sending a MIME message" on page 517.
- To configure the SMTP server, port number, and security identity for the EmailOutput node as a broker external resource, see "Creating an SMTP broker external resource" on page 520.

Sending an e-mail

You can send an e-mail with a static subject and static text to a static list of recipients.

Use the workbench to configure the properties on the EmailOutput node so that you can send an e-mail with a statically defined subject and text, and no attachments, to a statically defined list of recipients. The same e-mail is sent to the same recipients. This method is useful when you want to test the EmailOutput node, or when notification alone is sufficient.

1. Switch to the Broker Application Development perspective.
2. Add an EmailOutput node to your message flow.
3. Edit the following properties of the EmailOutput node.

a. On the **Basic** tab, add the SMTP Server and Port information (for example, *smtp.server.com:25*).
b. On the **Email** tab, add the e-mail addresses of recipients by using the To Addresses, Cc Addresses, and Bcc Addresses properties.
c. On the **Email** tab, add the e-mail address of the sender by using the From Address and Reply-To Address properties.
d. On the **Email** tab, provide a subject for the e-mail by using the Subject of email property.
e. On the **Email** tab, provide the text of the e-mail by using Email message text property.

4. Save the changes.
5. Add the message flow to the BAR file and deploy.

When a message is passed into the deployed EmailOutput node, an e-mail is sent to the defined set of recipients, containing the subject and text specified on the node.

Sending an e-mail with an attachment

You can send an e-mail with a fixed subject and fixed text, and an attachment, to a static list of recipients.

Use the workbench to configure the properties on the EmailOutput node so that you can send an e-mail with a statically defined subject and text, and an attachment, to a statically defined list of recipients. This method causes the e-mail message to be constructed as a MIME message. The subject, text, and list of recipients remains static, but the content of the attachment is sought dynamically from the message that is passed to the EmailOutput node at run time. The location of the attachment in the message is defined statically.

1. Switch to the Broker Application Development perspective.
2. Add an EmailOutput node to your message flow.
3. Edit the following properties of the EmailOutput node.
   a. On the **Basic** tab, add the SMTP Server and Port information (for example, *smtp.server.com:25*).
   b. On the **Email** tab, add the e-mail addresses of recipients by using the To Addresses, Cc Addresses, and Bcc Addresses properties.
   c. On the **Email** tab, add the e-mail address of the sender by using the From Address and Reply-To Address properties.
   d. On the **Email** tab, provide a subject for the e-mail by using the Subject of email property.
   e. On the **Email** tab, provide the text of the e-mail by using Email message text property.
   f. On the **Attachment** tab, set the Attachment Content property by using either an ESQL or XPath expression, referring to an element in the message tree; for example, *Body.BLOB*.
   g. On the **Attachment** tab, set the Attachment Content Name property with the name of the attachment as it appears in the e-mail.
   h. On the **Attachment** tab, set the Attachment Content Type, Attachment Content Encoding, and Multipart Content Type properties to determine the type of attachment that is sent in the MIME message.
4. Save the changes.
5. Add the message flow to the BAR file and deploy.

When a message is passed into the deployed EmailOutput node, an e-mail is sent to the defined set of recipients, containing the subject, text, and attachment specified on the node.

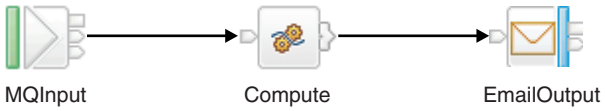
Producing dynamic e-mail messages

You can produce an e-mail where the SMTP server, list of recipients, subject, text, and multiple attachments are all determined at run time.

The node properties that you set when sending an e-mail can be optional, and can be overridden at run time by values that you specify in the local environment, e-mail output header (EmailOutputHeader), or the body of the message. To use this method, previous nodes in the message flow must construct these overrides. Where a text value is not specified in the node properties for the main body of the e-mail, the body of the message that is passed to the EmailOutput node is used.

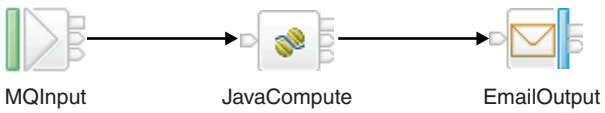
The following examples show how to set up the recipient, sender, subject, SMTP server, and message body information in ESQL (with a Compute node) and Java (with a JavaCompute node).

Using a Compute node




```
CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  CALL CopyMessageHeaders();

  -- Add recipient information to the EmailOutputHeader
  SET OutputRoot.EmailOutputHeader.To = '<recipient email address>';
  SET OutputRoot.EmailOutputHeader.Cc = '<recipient email address>';
  SET OutputRoot.EmailOutputHeader.Bcc = '<recipient email address>';

  -- Add sender information to EmailOutputHeader
  SET OutputRoot.EmailOutputHeader.From = '<sender email address>';
  SET OutputRoot.EmailOutputHeader."Reply-To" = '<reply email address>';

  -- Add subject to EmailOutputHeader
  SET OutputRoot.EmailOutputHeader.Subject = 'Replaced by ESQL compute node.';

  -- Add SMTP server information to the LocalEnvironment
  SET OutputLocalEnvironment.Destination.Email.SMTPServer ='<smtp.server:port>';

  -- Create a new message body, which will be sent as the main text of the email.
  SET OutputRoot.BLOB.BLOB = CAST('This is the new text for the body of the email.' AS BLOB CCSID 1208);

  RETURN TRUE;
 END;
```

Using a JavaCompute node



```
public void evaluate(MbMessageAssembly assembly) throws MbException {
  MbOutputTerminal out = getOutputTerminal("out");

  // Create a new assembly to propagate out of this node, as we want to update it
  MbMessage outMessage = new MbMessage();
  copyMessageHeaders(assembly.getMessage(), outMessage);
  MbMessage outLocalEnv = new MbMessage(assembly.getLocalEnvironment());
  MbMessage outExceptionList = new MbMessage(assembly.getExceptionList());
  MbMessageAssembly outAssembly = new MbMessageAssembly(assembly, outLocalEnv, outExceptionList, outMessage);
  MbElement localEnv = outAssembly.getLocalEnvironment().getRootElement();

  // Create the EmailOutputHeader parser. This is where we add recipient, sender and subject information.
  MbElement root = outMessage.getRootElement();
  MbElement SMTPOutput = root.createElementAsLastChild("EmailOutputHeader");

  // Add recipient information to EmailOutputHeader
  SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "To", "<recipient email address");
  SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Cc", "<recipient email address");
  SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Bcc", "<recipient email address");

  // Add sender information to EmailOutputHeader
  SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "From", "<sender email address>");
  SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Reply-To", "<reply email address>");

  // Add subject information to EmailOutputHeader
  SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Subject", "Replaced by Java compute node.");

  // Create Destination.Email. This is where we add SMTP server information
  MbElement Destination = localEnv.createElementAsLastChild(MbElement.TYPE_NAME, "Destination", null);
  MbElement destinationEmail = Destination.createElementAsLastChild(MbElement.TYPE_NAME, "Email", null);
  destinationEmail.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "SMTPServer", "<smtp.server:port>");

  // Set last child of root (message body)
  MbElement BLOB = root.createElementAsLastChild(MbBLOB.PARSER_NAME);
  String text = "This is the new text for the body of the email";
  BLOB.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "BLOB", text.getBytes());

  outMessage.finalizeMessage(MbMessage.FINALIZE_VALIDATE);

  out.propagate(outAssembly);  }
```

Using the local environment

Use the local environment to specify overrides to the SMTP server connection information and attachments.

Local environment	Description
Destination.Email.SMTPServer	The Server:Port of the SMTP server. Port is optional; if you do not specify it, the default value is 25.
Destination.Email.SecurityIdentity	The security identity for authentication with the SMTP server, which can be the name of the userid and password pair that is defined using the mqsisetdbparms command, or it can reference an external resource that has a securityIdentity attribute that references a userid and password that are defined using the mqsisetdbparms command. In both cases, the value is appended after the string “smtp::”. For example, if you use the mqsisetdbparms command to create a userid and password of *smtp::myUseridPassword*, the securityIdentity that is specified on the node, or indirectly in an external resource, is *myUseridPassword*.
Destination.Email.BodyContentType	Identifies that the body of the e-mail message contains HTML rather than plain text. You can set this property to text/plain, text/html, or text/xml; text/plain is the default value.

Local environment	Description
Destination.Email.MultiPartContentType	The type of multipart, including related, mixed, and alternative. You can set any value here.
Destination.Email.Attachment.Content	Either the actual attachment (BLOB/text), or an XPath or ESQL expression that references an element; for example, an element in the message tree or LocalEnvironment. The value of the referenced element is taken as the content of the attachment. • If the element is a BLOB, it is an attachment. • If the element is text, check to see if it can be resolved to another element in the message tree or LocalEnvironment. If it can be resolved, use that element. If it cannot be resolved, add this element as the attachment.
Destination.Email.Attachment.ContentType	The type of attachment (also known as Internet Media Type), including text/plain, text/html, and text/xml. You can set any value here.
Destination.Email.Attachment.ContentName	The name of the attachment.
Destination.Email.Attachment.ContentEncoding	The encoding of the attachment: 7bit, base64, or quoted-printable. • 7bit is the default value that is used for ASCII text. • Base64 is used for non ASCII, whether non English or binary data. This format can be difficult to read. • Quoted-printable is an alternative to Base64, and is appropriate when most of the data is ASCII with some non-ASCII parts. This format is more readable; it provides a more compact encoding because the ASCII parts are not encoded.

Using the e-mail output header

Use the e-mail output header to specify overrides to the SMTP server connection information and attachments. The EmailOutputHeader is a child of Root. Values that you specify in this header override equivalent properties that you set on the EmailOutput node. Use the SMTP output header to specify any of the e-mail attributes, such as its recipients.

Location	Description
Root.EmailOutputHeader.To	A comma-separated list of e-mail addresses.
Root.EmailOutputHeader.Cc	A comma-separated list of e-mail addresses.
Root.EmailOutputHeader.Bcc	A comma-separated list of e-mail addresses.
Root.EmailOutputHeader.From	A comma-separated list of e-mail addresses.
Root.EmailOutputHeader.Reply-To	A comma-separated list of e-mail addresses.
Root.EmailOutputHeader.Subject	The subject of the e-mail.

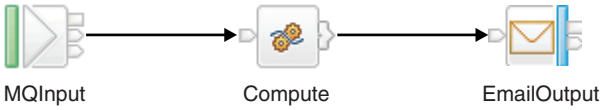
Sending a MIME message

You can send an e-mail that is constructed from a MIME message.

You can pass a MIME message to the EmailOutput node, which uses the MIME parser to write the MIME message to a bit stream. This message is then sent to the list of recipients in the EmailOutputHeader. Local environment overrides are not considered when a MIME message is passed. You do not need to configure the EmailOutput node in the following examples.

The following examples show how to set up the recipient, sender, subject, SMTP server, and message body information in ESQL (with a Compute node) and Java (with a JavaCompute node).

Using a Compute node




```
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN

   CALL CopyMessageHeaders();

   -- Add recipient information to the EmailOutputHeader
   SET OutputRoot.EmailOutputHeader.To = '<recipient e-mail address>';
   SET OutputRoot.EmailOutputHeader.Cc = '<recipient e-mail address>';
   SET OutputRoot.EmailOutputHeader.Bcc = '<recipient e-mail address>';

   -- Add sender information to EmailOutputHeader
   SET OutputRoot.EmailOutputHeader.From = '<sender e-mail address>';
   SET OutputRoot.EmailOutputHeader."Reply-To" = '<reply e-mail address>';

   -- Add subject to EmailOutputHeader
   SET OutputRoot.EmailOutputHeader.Subject = 'Dynamic MIME message in ESQL.';

   -- Add SMTP server information to the LocalEnvironment
   SET OutputLocalEnvironment.Destination.Email.SMTPServer ='<smtp.server:port>';

   -- Create a new MIME message body, which will be sent as the main text of the e-mail,
   -- including an attachment.
   CREATE FIELD OutputRoot.MIME TYPE Name;
   DECLARE M REFERENCE TO OutputRoot.MIME;

   -- Create the Content-Type child of MIME explicitly to ensure the correct order. If we set
   -- the ContentType property instead, the field could appear as the last child of MIME.
   CREATE FIELD M."Content-Type" TYPE NameValue VALUE 'multipart/related; boundary=myBoundary';
   CREATE FIELD M."Content-ID"   TYPE NameValue VALUE 'new MIME document';

   CREATE LASTCHILD OF M TYPE Name NAME 'Parts';
   CREATE LASTCHILD OF M.Parts TYPE Name NAME 'Part';
   DECLARE P1 REFERENCE TO M.Parts.Part;

   -- First part:
   --   Create the body of the e-mail.
   --   The body of the e-mail has the text 'This is the main body of the e-mail.'.
   CREATE FIELD P1."Content-Type" TYPE NameValue VALUE 'text/plain; charset=us-ascii';
   CREATE FIELD P1."Content-Transfer-Encoding" TYPE NameValue VALUE '8bit';
   CREATE LASTCHILD OF P1 TYPE Name NAME 'Data';
   CREATE LASTCHILD OF P1.Data DOMAIN('BLOB') PARSE(CAST('This is the main body of the e-mail.'
AS BLOB CCSID 1208));

   CREATE LASTCHILD OF M.Parts TYPE Name NAME 'Part';
   DECLARE P2 REFERENCE TO M.Parts.Part[2];

   -- Second part:
   --   Create the attachment of an e-mail.
   --   The attachment is called 'attachment.txt' and contains the text 'This is an attachment.'.
   CREATE FIELD P2."Content-Type" TYPE NameValue VALUE 'text/plain; charset=us-ascii; name=attachment.txt';
   CREATE FIELD P2."Content-Transfer-Encoding" TYPE NameValue VALUE '8bit';
   CREATE LASTCHILD OF P2 TYPE Name NAME 'Data';
```

```
    CREATE LASTCHILD OF P2.Data DOMAIN('BLOB') PARSE(CAST('This is an attachment.' AS BLOB CCSID 1208));

    RETURN TRUE;
END;
```

Using a JavaCompute node


```
public void evaluate(MbMessageAssembly assembly) throws MbException {
    MbOutputTerminal out = getOutputTerminal("out");
    MbOutputTerminal fail = getOutputTerminal("fail");

    // Create a new assembly to propagate out of this node, as we want to
    // update it
    MbMessage outMessage = new MbMessage();
    copyMessageHeaders(assembly.getMessage(), outMessage);
    MbMessage outLocalEnv = new MbMessage(assembly.getLocalEnvironment());
    MbMessage outExceptionList = new MbMessage(assembly.getExceptionList());
    MbMessageAssembly outAssembly = new MbMessageAssembly(assembly, outLocalEnv, outExceptionList, outMessage);
    MbElement localEnv = outAssembly.getLocalEnvironment().getRootElement();

    // Create the EmailOutputHeader parser. This is where we add recipient,
    // sender and subject information.
    MbElement root = outMessage.getRootElement();
    MbElement SMTPOutput = root.createElementAsLastChild("EmailOutputHeader");

    // Add recipient information to EmailOutputHeader
    SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "To", "<recipient e-mail address>");
    SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Cc", "<recipient e-mail address>");
    SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Bcc", "<recipient e-mail address>");

    // Add sender information to EmailOutputHeader
    SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "From", "<sender e-mail address>");
    SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Reply-To", "<reply e-mail address>");

    // Add subject information to EmailOutputHeader
    SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Subject", "Dynamic MIME message in Java.");

    // Create Destination.Email. This is where we add SMTP server information.
    MbElement Destination = localEnv.createElementAsLastChild(MbElement.TYPE_NAME, "Destination", null);
    MbElement destinationEmail = Destination.createElementAsLastChild(MbElement.TYPE_NAME, "Email", null);
    destinationEmail.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "SMTPServer", "<smtp.server:port>");

    // Set last child of root (message body) as MIME.
    MbElement MIME = root.createElementAsLastChild("MIME");

    // Create the Content-Type child of MIME explicitly to ensure the correct order.
    MIME.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Content-Type", "multipart/related;
boundary=myBoundary");
    MIME.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Content-ID", "new MIME document");
    MbElement parts = MIME.createElementAsLastChild(MbElement.TYPE_NAME, "Parts", null);
    MbElement part, data, blob;
    String text;

    // First part:
    //   Create the body of the e-mail.
    //   The body of the e-mail has the text 'This is the main body of the e-mail.'.
    part = parts.createElementAsLastChild(MbElement.TYPE_NAME, "Part", null);
    part.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Content-Type", "text/plain; charset=us-ascii");
    part.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Content-Transfer-Encoding", "8bit");
    data = part.createElementAsLastChild(MbElement.TYPE_NAME, "Data", null);
```

```
    blob = data.createElementAsLastChild("BLOB");
    text = "This is the main body of the e-mail.";
    try {
   blob.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "BLOB", text.getBytes("UTF8"));
    } catch (UnsupportedEncodingException e) {
   fail.propagate(outAssembly);
    }

    // Second part:
    //   Create the attachment of an e-mail.
    //   The attachment is called 'attachment.txt' and contains the text 'This is an attachment.'.
    part = parts.createElementAsLastChild(MbElement.TYPE_NAME, "Part", null);
    part.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Content-Type", "text/plain; charset=us-ascii;
name=attachment.txt");
    part.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Content-Transfer-Encoding", "8bit");
    data = part.createElementAsLastChild(MbElement.TYPE_NAME, "Data", null);
    blob = data.createElementAsLastChild("BLOB");
    text = "This is an attachment.";
    try {
   blob.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "BLOB", text.getBytes("UTF8"));
    } catch (UnsupportedEncodingException e) {
   fail.propagate(outAssembly);
    }

    outMessage.finalizeMessage(MbMessage.FINALIZE_VALIDATE);
    out.propagate(outAssembly);
}
```

Creating an SMTP broker external resource

You can configure the SMTP server, port number, and security identity for the EmailOutput node as a broker external resource.

Use an alias that is specified in the SMTP Server and Port property on the EmailOutput node. The security identity references a user ID and password pair that is defined on the broker by using the mqsisetdbparms command.

1. Use the mqsicreateconfigurableservice command to create an SMTP broker external resource for the alias that is specified on the node.
2. Then use the mqsichangeproperties command to create an SMTPServer property with the value in the form of *server:port*. The port value is optional; if you do not specify it, the default value is 25. You can also use the mqsichangeproperties command to create an SMTPSecurityIdentity property with a value that is the name of a security identity that can be resolved at run time to a user ID and password for authentication with the SMTP server.

   For example:

```
mqsicreateconfigurableservice MY_BROKER –c SMTP –o SMTP_MyAlias
```

   followed by:

```
mqsichangeproperties MY_BROKER –c SMTP –o SMTP_MyAlias –n serverName –v smtp.hursley.ibm.com:25
```

   These commands override the SMTP server and port values that are specified on any nodes that also specify an alias of SMTP_MyAlias. If the local environment contains any overrides, they take preference over the broker external resource properties. See the following example:

```
mqsichangeproperties MY_BROKER –c SMTP –o SMTP_MyAlias –n securityIdentity –v mySecurityIdentity
```

   You must also use the mqsisetdbparms command to define the security identity at the broker run time.

Developing Java

When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

To tailor the behavior of each node, create a Java class file that provides the processing that you want. You manage Java files through the Java perspective.

You can add any valid Java code to a JavaCompute node, making full use of the Java user-defined node API to process an incoming message. You can use the Java editing facilities of the Eclipse platform to develop your Java code. These facilities include:

- Code completion
- Integrated Javadoc documentation
- Automatic compilation

The Java user-defined node API includes some extra methods that simplify tasks that involve message routing and transformation. These tasks include accessing named elements in a message tree, setting their values, and creating elements, without the need to navigate the tree explicitly.

Use the Debug perspective to debug a message flow that contains a JavaCompute node. When control passes to a JavaCompute node during debugging, the perspective opens the Java debugger, and you can step through the Java class code for the node.

This section provides the following information on developing Java:

- "Managing Java Files"
- "Writing Java" on page 526

Managing Java Files

The Java code that you provide to modify or customize the behavior of a JavaCompute node is stored in a Java project. WebSphere Message Broker uses the Eclipse Java perspective for developing and administering Java files.

This section contains topics that describe how to manage these files:

- "Creating Java code for a JavaCompute node"
- "Opening an existing Java file" on page 523
- "Saving a Java file" on page 523
- "Adding Java code dependencies" on page 524
- "Deploying JavaCompute node code" on page 525

Creating Java code for a JavaCompute node

Use these instructions to associate Java code with your JavaCompute node.

Before you start

To complete this task, you must have already created a JavaCompute node in your message flow.

To associate Java code with a JavaCompute node, use one of the following methods:

- Use the New Java Compute Node Class wizard to create template code. This is a preferred method.
  1. Right-click the node and click **Open Java**.
  2. Navigate the New Java Compute Node Class wizard until you reach the Java Compute Node Class Template page. On the Java Compute Node Class Template page, choose one of the following options:
     - For a filter node template code, choose **Filtering message class**.
     - To change an incoming message, choose **Modifying message class**.
     - To create a new message, choose **Creating message class**.

  You have created template code for your JavaCompute node.
- Associate a JavaCompute node with an existing Java class that the wizard has previously generated; this is the safest way in which you can share the same Java code between multiple nodes. To associate a JavaCompute nodes with an existing Java class, perform the following steps:
  1. Right-click the JavaCompute node and click **Properties**.
  2. Enter the name of the Java class in the **Java Class** field.
  3. Click **OK**.

  You have associated your JavaCompute node with an existing Java class.
- Create a Java project from scratch. Before you add one or more classes to the project, you must perform the following steps:
  1. Open the `.project` file in the text editor, and ensure that the following builders and natures are set:

```
<buildSpec>
  <buildCommand>
   <name>org.eclipse.jdt.core.javabuilder</name>
   <arguments>
   </arguments>
  </buildCommand>
  <buildCommand>
   <name>com.ibm.etools.mft.java.builder.javabuilder</name>
   <arguments>
   </arguments>
  </buildCommand>
  <buildCommand>
   <name>com.ibm.etools.mft.jcn.jcnbuilder</name>
   <arguments>
   </arguments>
  </buildCommand>
  <buildCommand>
   <name>com.ibm.etools.mft.bar.barbuilder</name>
   <arguments>
   </arguments>
  </buildCommand>
 </buildSpec>
 <natures>
  <nature>org.eclipse.jdt.core.javanature</nature>
  <nature>com.ibm.etools.mft.bar.barnature</nature>
  <nature>com.ibm.etools.mft.jcn.jcnnature</nature>
 </natures>
```

  2. Add the following plug-ins to the build path of the Java project:
     a. Open the properties of the Java project.
     b. Select **Java Build Path** and open the **Libraries** tab.
     c. Click **Add Variable**.
     d. Select the variable `JCN_HOME` and click **OK**.

e. Double-click the variable you added to open the **Edit Variable Entry** dialog.
f. Click **Extension** and select `javacompute.jar`.
g. Repeat the previous four steps to add the variable `JCN_HOME/jplugin2.jar`.

3. Create the appropriate Java class and ensure that it extends from `com.ibm.broker.javacompute.MbJavaComputeNode`.

You have created your Java project.

You can now perform the following tasks:

- "Opening an existing Java file"
- "Saving a Java file"
- "Adding Java code dependencies" on page 524

Opening an existing Java file

You can add to and modify Java code that you have created in a Java project.

Before you start

Before you start this task, complete the following tasks:
- Add a "JavaCompute node" on page 1013 to your message flow.
- "Creating Java code for a JavaCompute node" on page 521

To open an existing Java file:

1. Switch to the Java perspective.
2. In the Package Explorer view, double-click the Java file that you want to open. The file is opened in the editor view.
3. Work with the contents of the file to make your changes.

You can also open a Java file when you have a message flow open in the editor view. Right-click the JavaCompute node, then select **Open Java**.

Next:

You can now perform the following tasks:

- "Saving a Java file"
- "Adding Java code dependencies" on page 524

Saving a Java file

When you edit your Java files, save them to preserve the additions and modifications that you have made.

Before you start

To complete this task, you must have completed the following tasks:
- Add a "JavaCompute node" on page 1013 to your message flow.
- "Creating Java code for a JavaCompute node" on page 521

To save a Java file:

1. Switch to the Java perspective.
2. Create a new Java file or open an existing Java file.
3. Make the changes to the contents of the Java file.

4. When you have finished working, click **File** → **Save** or **File** → **Save All** to save the file and retain all your changes.

Next:

You can now perform the following task:

- "Adding Java code dependencies"

Adding Java code dependencies

When you write Java code for a JavaCompute node, you can include references to other Java projects and JAR files.

Before you start

To complete this task, you must have completed the following tasks:

- Add a "JavaCompute node" on page 1013 to your message flow.
- Create Java code for a JavaCompute node.

The Java code in a JavaCompute node might contain references to other Java projects in the Eclipse workspace (internal dependencies), or to external JAR files, for example the JavaMail API (external dependencies). If other JAR files are referenced, you must add the files to the project class path.

1. Right-click the project folder for the project on which you are working, and click **Properties**.
2. Click **Java Build Path** in the left pane.
3. Click the **Libraries** tab.
4. Complete one of the following steps:
   - To add an internal dependency, click **Add JARs**, select the JAR file that you want to add, then click **OK**.
   - To add an external dependency, click **Add External JARs**, select the JAR file that you want to add, then click **Open**. Copy the file to *WorkPath*`/shared-classes` where *WorkPath* is the full path to the working directory of the broker. If you do not copy the external dependencies here, `ClassNotFoundException` exceptions are generated at run time.

   Tip:

   The default value for *WorkPath* depends on your operating system:
   - On Linux and UNIX systems: `/var/mqsi/common/profiles`
   - The default working directory is `%ALLUSERSPROFILE%\Application Data\IBM\MQSI\common\profiles`, where `%ALLUSERSPROFILE%` is the environment variable that defines the system working directory. The default directory depends on the operating system:
     - On Windows XP and Windows Server 2003: `C:\Documents and Settings\All Users\Application Data\IBM\MQSI\common\profiles`
     - On Windows Vista and Windows Server 2008: `C:\ProgramData\IBM\MQSI\common\profiles`

     The value itself might be different on your computer.

You have now added a code dependency.

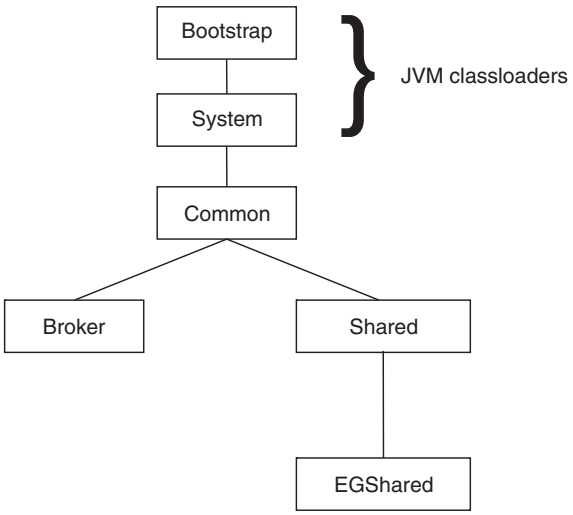
Deploying JavaCompute node code

The Message Broker Toolkit handles the deploying of JavaCompute node code automatically. When you create a BAR file and add the message flow, the Message Broker Toolkit packages the compiled Java code and its dependencies into the BAR file.

JavaCompute node classloading

When you include one or more JavaCompute nodes in a broker archive (BAR) file, the JAR files are loaded in a separate classloader. The classloader loads all classes that are packaged in the deployed BAR file. These classes override any classes that are in the shared classes directory or the CLASSPATH environment variable.

The broker uses the following classloader tree:



These components are in the classloader tree:

- **Common classloader**: loads the classes that are shared between the broker and user code. For example, the classes that are contained in `jplugin2.jar` are common to the broker and the user code.
- **Broker classloader**: loads the broker internal classes. These classes cannot be accessed by user classes.
- **Shared classloader**: loads classes from JAR files that have been placed in the *workpath*`/shared-classes/` directory, and from the CLASSPATH environment variable. These classes are available to all Java user-defined nodes and JavaCompute nodes in the broker.

  The CLASSPATH environment variable can contain the wildcard character (*) at the end of a directory path specifier. The wildcard is expanded to include all files in that directory with the extension `.jar` or `.JAR`.

  You can find `SharedClassLoader.getInstance()` in the `bootstrap.jar` file, which is in the classpath of the run time; to use it, you must add it to your Eclipse project.
- **EGShared classloader**: loads all classes that are deployed to the execution group in the broker archive (BAR) file, either by a JavaCompute node or an ESQL-to-Java mapping.

This method is used to support the deployment mechanism for the JavaCompute node. Each time a BAR file is deployed, a new instance of the EGShared classloader is created and the old instance is discarded. This action allows the JavaCompute node to reload modified versions of the same class without the need to restart the broker.

The broker uses the following search path to find JavaCompute node classes:

1. The deployed JAR file
2. `<WorkPath>/shared-classes/` to locate any JAR files
3. The `CLASSPATH` environment variable

Writing Java

When you create a message flow, you include input nodes that receive messages and, optionally, output nodes that send out new or updated messages. If the processing that must be performed on the message requires it, you can include other nodes after the input node that are customized in Java to complete the actions that your applications need.

Some of the built-in nodes allow you to customize the processing that they provide. In a JavaCompute node, you can provide Java code that controls precisely the behavior of the node. This set of topics discusses how you can use Java to customize the JavaCompute node.

Using a JavaCompute node you can check and manipulate message content. You can:

- Read the contents of the input message
- Construct new output messages that are created from all, part, or none of the input message

Use the Debug perspective to debug a message flow that contains a JavaCompute node. When control passes to a JavaCompute node during debugging, the perspective opens the Java debugger, allowing you to step through the Java class code for the node.

This section provides more information about writing Java:

- Manipulating message body data
- Manipulating other parts of the message tree
- Accessing broker properties
- Accessing user-defined properties
- “Adding keywords to JAR files” on page 540
- Interacting with databases
- “Calling an Enterprise Java Bean” on page 544
- Handling exceptions
- Logging errors

Manipulating message body data by using a JavaCompute node

You can manipulate message body data by using a JavaCompute node.

The message body is always the last child of root, and its parser name identifies it, for example XML or MRM.

The following topics describe how to refer to, modify, and create message body data. The information provided here is domain independent:

- "Accessing elements in a message tree from a JavaCompute node"
- "Transforming a message by using a JavaCompute node" on page 529
- "Creating a simple filter by using a JavaCompute node" on page 532
- "Propagating a message to the JavaCompute node Out and Alternate terminals" on page 533
- "Extracting information from a message by using XPath 1.0 and a JavaCompute node" on page 533

Accessing elements in a message tree from a JavaCompute node:

Access the contents of a message, for reading or writing, using the structure and arrangement of the elements in the tree that the parser creates from the input bit stream.

Follow the relevant parent and child relationships from the top of the tree downwards until you reach the required element.

The message tree is passed to a JavaCompute node as an argument of the evaluate method. The argument is an MbMessageAssembly object. MbMessageAssembly contains four message objects:

- Message
- Local Environment
- Global Environment
- Exception List

These objects are read-only, except for Global Environment. If you try to write to the read-only objects, an MbReadOnlyException is issued.

This topic contains the following information about accessing elements in a message tree:

- "Traversing the element tree"
- "Accessing information about an element" on page 529

Traversing the element tree:

The following table shows the Java methods that you can use to access element trees, and the equivalent ESQL field type constant for each point in the tree.

Java accessor from MbMessageAssembly	ESQL field type constant
getMessage().getRootElement()	InputRoot
getMessage().getRootElement().getLastChild()	InputBody
getLocalEnvironment().getRootElement()	InputLocalEnvironment
getGlobalEnvironment().getRootElement()	Environment
getExceptionList().getRootElement()	InputExceptionList

Use the following methods to traverse a message tree from an element of type MbElement:

getParent()
returns the parent of the current element

getPreviousSibling()
returns the previous sibling of the current element

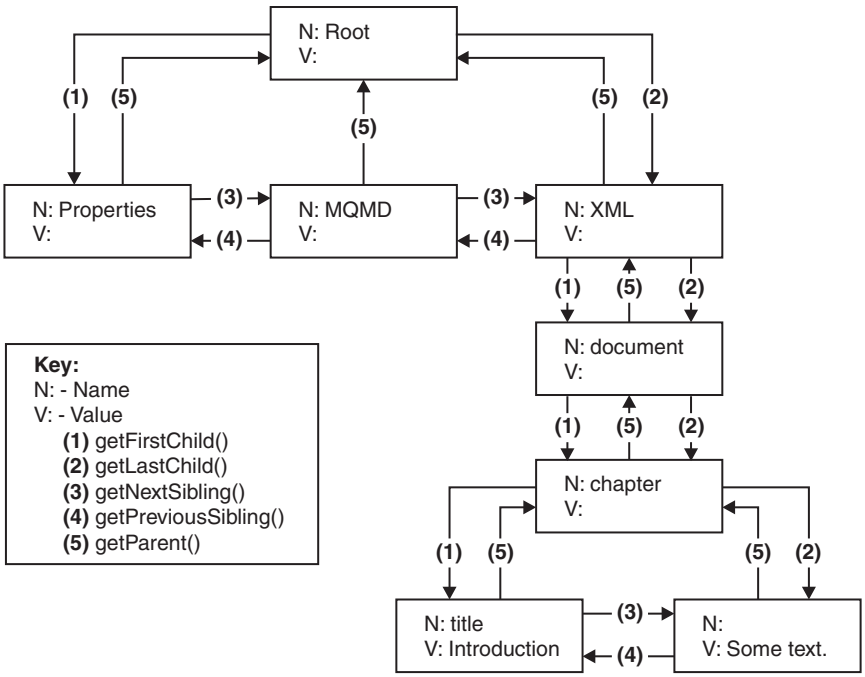
getNextSibling()
returns the next sibling of the current element

getFirstChild()
returns the first child of the current element

getLastChild()
returns the last child of the current element

The following example shows a simple XML message and the logical tree that would be created from the message. The message has been sent using WebSphere MQ. The logical tree diagram also shows the methods to call in order to navigate around the tree.

```
<document>
  <chapter title='Introduction'>
    Some text
  </chapter>
</document>
```



The tree used in this diagram is the one that is created by parsing the previous XML example.

- From the Root part of the tree, calling getFirstChild() navigates to Properties. Also from Root, calling getLastChild() returns XML.
- From Properties, calling getParent() returns Root, and calling getNextSibling() returns MQMD.
- From MQMD, calling getPreviousSibling() returns Properties, calling getParent() returns Root, and calling getNextSibling() returns XML.
- From XML, calling getPreviousSibling() returns MQMD, calling getParent() returns Root, calling getFirstChild() returns document, and calling getLastChild() also returns document.

- From document, calling getParent() returns XML, calling getFirstChild() returns chapter, and calling getLastChild() also returns chapter.
- From chapter, calling getParent() returns document, calling getFirstChild() returns title, and calling getLastChild() returns the child that contains the message data "Some text.".

The following Java code accesses the chapter element in the logical tree for an XML message that does not contain white spaces. The XML parser retains white space in the parsed tree, but the XMLNS and XMLNSC parsers do not.

```
MbElement root = assembly.getMessage().getRootElement();
MbElement chapter = root.getLastChild().getFirstChild().getFirstChild();
```

Accessing information about an element:

Use the following methods to return information about the referenced element:

getName()
returns the element name as a java.lang.String

getValue()
returns the element value

getType()
returns the generic type, which is one of the following types:
- NAME: an element of this type has a name, but no value.
- VALUE: an element of this type has a value, but no name.
- NAME/VALUE: an element of this type has both a value and a name.

getSpecificType()
returns the parser-specific type of the element

getNamespace()
returns the namespace URI of this element

Transforming a message by using a JavaCompute node:

You can transform a message by using a JavaCompute node.

The topics in this section describe how to transform messages by using a JavaCompute node:
- "Creating a new message by using a JavaCompute node"
- "Copying a message by using a JavaCompute node" on page 530
- "Setting, copying, and moving message elements by using a JavaCompute node" on page 530
- "Creating new elements by using a JavaCompute node" on page 531

Creating a new message by using a JavaCompute node:

Many message transformation scenarios require a new outgoing message to be built. The *Create Message Class* template in the JavaCompute node wizard generates template code for this.

In the template code, the default constructor of MbMessage is called to create a blank message, as shown in the following Java code:

```
MbMessage outMessage = new MbMessage();
```

The headers can be copied from the incoming message by using the supplied utility method, copyMessageHeaders(), as shown in this Java code:

```
copyMessageHeaders(inMessage, outMessage);
```

The new message body can now be created. First, add the top level parser element. For XML, this is:

```
MbElement outRoot = outMessage.getRootElement();
MbElement outBody = outRoot.createElementAsLastChild(MbXMLNSC.PARSER_NAME);
```

The remainder of the message can then be built up by using the createElement methods and the extended syntax of the broker XPath implementation.

When you wish to create a BLOB message, that is handled as a single byte string by using the BLOB parser domain. The message data is added as a byte array to the single element named "BLOB" under the parser level element, described as follows:

```
String myMsg = "The Message Data";
MbElement outRoot = outMessage.getRootElement();
// Create the Broker Blob Parser element
MbElement outParser = outRoot.createElementAsLastChild(MbBLOB.PARSER_NAME);
// Create the BLOB element in the Blob parser domain with the required text
MbElement outBody = outParser.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "BLOB", myMsg.getBytes());
```

You can use the following code to create a BLOB message in a JavaCompute node:

```
MbElement outMsgRootEl = outMessage.getRootElement();

 String parserName = MbBLOB.PARSER_NAME;
          String messageType = "";
          String messageSet = "";
          String messageFormat = "";
          int encoding = 0;
          int ccsid = 0;
          int options = 0;
 outMsgRootEl.createElementAsLastChildFromBitstream(responseBodyXmlData,
   parserName, messageType, messageSet, messageFormat, encoding, ccsid,
         options);
```

Copying a message by using a JavaCompute node:

You can copy a message by using a JavaCompute node.

The incoming message and message assembly are read-only. In order to modify a message, a copy of the incoming message must be made. The *Modifying Message Class* template in the JavaCompute node wizard generates this copy. The following copy constructors are called:

```
MbMessage outMessage = new MbMessage(inAssembly.getMessage);
MbMessageAssembly outAssembly = new MbMessageAssembly(inAssembly, outMessage);
```

The new outAssembly object is propagated to the next node.

Setting, copying, and moving message elements by using a JavaCompute node:

Transform elements in the message as it passes through a JavaCompute node in the message flow.

- "Setting information about an element" on page 531
- "Moving and copying elements" on page 531

The Java API reference information provides details about each of the methods used in the following sections:

Setting information about an element:

Use these methods to set information about the referenced element:

setName()
Sets the name of the element

setValue()
Sets the value of the element

setSpecificType()
Sets the parser-specific type of the element

setNamespace()
Sets the namespace URI of the element

Moving and copying elements:

Use a JavaCompute node to copy or detach an element from a message tree by using the following methods:

detach()
The element is detached from its parent and siblings, but any child elements are left attached

copy() A copy of the element and its attached children is created

Use one of four methods to attach an element or subtree that you have copied on to another tree:

addAsFirstChild(*element***)**
Adds an unattached element as the first child of *element*

addAsLastChild(*element***)**
Adds an unattached element as the last child of *element*

addBefore(*element***)**
Adds an unattached element as the previous sibling of *element*

addAfter(*element***)**
Adds an unattached element as the next sibling of *element*

Creating new elements by using a JavaCompute node:

You can use a JavaCompute node to create new elements.

Use the following methods in a JavaCompute node to create new elements in a message tree:

- createElementAsFirstChild()
- createElementAsLastChild()
- createElementBefore()
- createElementAfter()

The method returns a reference to the newly-created element. Each method has three overloaded forms:

createElement...(int type)
Creates a blank element of the specified type. Valid generic types are:

- MbElement.TYPE_NAME. This type of element has only a name, for example an XML element.
- MbElement.TYPE_VALUE. This type of element has only a value, for example XML text that is not contained in an XML element.
- MbElement.TYPE_NAME_VALUE. This type of element has both a name and a value, for example an XML attribute.

Specific type values can also be assigned. The meaning of this type information is dependent on the parser. Element name and value information must be assigned by using the setName() and setValue() methods.

createElement...(int type, String name, Object value)
Method for setting the name and value of the element at creation time.

createElement...(String parserName)
A special form of createElement...() that is only used to create top-level parser elements.

This example Java code adds a new chapter element to the XML example given in "Accessing elements in a message tree from a JavaCompute node" on page 527:

```
MbElement root = outMessage.getRootElement();
MbElement document = root.getLastChild().getFirstChild();
MbElement chapter2 = document.createElementAsLastChild(MbElement.TYPE_NAME,"Chapter",null);

// add title attribute
MbElement title2 = chapter2.createElementAsFirstChild(MbElement.TYPE_NAME_VALUE,
 "title", "Message Flows");
```

This produces the following XML output:

```
<document>
 <chapter title="Introduction">
  Some text.
 </chapter>
 <chapter title="Message Flows"/>
</document>
```

Creating a simple filter by using a JavaCompute node:

The JavaCompute node has two output terminals: Out and Alternate. To use the JavaCompute node as a filter node, propagate a message to either the Out or Alternate terminal based on the message content.

Before you start

To complete this task, you must have added a "JavaCompute node" on page 1013 to your message flow.

Use the JavaCompute node creation wizard to generate template code for a filter node:

Select the *Filtering Message Class* template in the JavaCompute node creation wizard to create a filter node.
The following template code is produced. It passes the input message to the Out terminal without doing any processing on the message.

```
public class jcn2 extends MbJavaComputeNode {

  public void evaluate(MbMessageAssembly assembly) throws MbException {
    MbOutputTerminal out = getOutputTerminal("out");
```

```
    MbOutputTerminal alt = getOutputTerminal("alternate");

    MbMessage message = assembly.getMessage();

    // ----------------------------------------------------------
    // Add user code below

    // End of user code
    // ----------------------------------------------------------

    // The following should only be changed
    // if not propagating message to the 'out' terminal

    out.propagate(assembly);
  }
}
```

The template produces a partial implementation of a method called evaluate(). The broker calls evaluate() once for each message that passes through the node. The parameter that is passed to evaluate() is the message assembly. The message assembly encapsulates the message that is passed on from the previous node in the message flow.
Add custom code to the template, and propagate messages to both the Out and Alternate terminals to create a message filter.

Propagating a message to the JavaCompute node Out and Alternate terminals:

The JavaCompute node has two output terminals, Out and Alternate. Therefore, you can use the node both as a filter node and as a message transformation node. After you have processed the message, propagate the message to an output terminal by using a propagate() method.

To propagate the message assembly to the Out terminal use the following method:

```
out.propagate(assembly);
```

To propagate the message assembly to the Alternate terminal, use the following method:

```
alt.propagate(assembly);
```

To propagate the same MbMessage object multiple times, call the finalizeMessage() method on the MBMessage object, so that any changes made to the message are reflected in the bit stream that is generated downstream of the JavaCompute node; for example:

```
MbMessage outMessage = new MbMessage(inMessage);
MbMessageAssembly outAssembly = new MbMessageAssembly(inAssembly, outMessage);
...
newMsg.finalizeMessage(MbMessage.FINALIZE_NONE);
out.propagate(outAssembly);
...
newMsg.finalizeMessage(MbMessage.FINALIZE_NONE);
out.propagate(outAssembly);
```

Extracting information from a message by using XPath 1.0 and a JavaCompute node:

XPath is a query language designed for use with XML documents, but you can use it with any tree structure to query contents.

WebSphere Message Broker uses XPath to select elements from the logical message tree regardless of the format of the bit stream. The terminology used in this topic is based on the terminology used in the W3C definition of XPath 1.0. For more information about XPath, see "Using XPath" on page 476; and for more information about the W3C definition of the XPath 1.0 standard, see W3C XPath 1.0 Specification. For examples of XPath use, see the MbXPath topic in the Java user-defined node API documentation.

This topic contains the following information:

Using the evaluateXPath method to extract message information

The evaluateXPath() method is included in the Java user-defined node API. It supports XPath 1.0, with the following exceptions:

- Namespace axis and namespace node type. The namespace axis returns the actual XML namespace declaration nodes for a particular element. You can therefore manipulate XML prefix or URI declarations within an XPath expression. This axis returns an empty node set for bit streams that are not XML.
- If you use the id() function, it throws an MbRecoverableException.

The evaluateXPath() method can be called on a MbMessage object (for absolute paths), or on a MbElement object (for relative paths). The XPath expression is passed to the method as a string parameter. A second form of this method is provided that takes an MbXPath object. This object encapsulates an XPath expression along with variable bindings and namespace mappings, if these are required.

The evaluateXPath() method returns an object of one of these four types, depending on the expression return type:

- java.lang.Boolean, representing the XPath Boolean type
- java.lang.Double, representing the XPath number type
- java.lang.String, representing the XPath string type
- java.util.List, representing the XPath node set. The List interface represents an ordered sequence of objects, in this case MbElements. It allows direct access to the elements, or the ability to get an Iterator or an MbElement array.

XPath variable binding

XPath 1.0 supports the ability to refer to variables that have been assigned before the expression that contains them is evaluated. The MbXPath class has three methods for assigning and removing these variable bindings from user Java code. The value must be one of the four XPath 1.0 supported types:

- Boolean
- node set
- number
- string

XPath namespace support

For XML messages, namespaces are referred to by using a mapping from an abbreviated namespace prefix to the full namespace URI, as shown in the following XML example:

```
<ns1:aaa xmlns:ns1='http://mydomain.com/namespace1'
         xmlns:ns2='http://mydomain.com/namespace2'>
  <ns2:aaa>
    <ns1:bbb/>
  </ns2:aaa>
</ns1:aaa>
```

The namespace prefix is convenient for representing the namespace, but is meaningful only within the document that defines that mapping. The namespace URI defines the global meaning. Also, the concept of a namespace prefix is not meaningful for documents that are generated in a message flow, because a namespace URI can be assigned to a syntax element without an XMLNS mapping having been defined.

For this reason, the XMLNSC and MRM parsers expose only the namespace URI to the broker and to user code (ESQL or user-defined code). By using ESQL, you can set up your own mappings to create abbreviations to these potentially long URIs. These mappings are not related in any way to the prefixes that are defined in the XML document (although they can be the same name).

By using the XPath processor you can map namespace abbreviations on to URIs that are expanded at evaluation time. The MbXPath class contains methods to assign and remove these namespace mappings. For example:

```
MbXPath xp = new MbXPath("/aaa/other:aaa/bbb");
xp.addNamespacePrefix("other", "http://mydomain.com/namespace2");
xp.setDefaultNamespace("http://mydomain.com/namespace2");
List nodeset = (List)message.evaluateXPath(xp);
```

Updating a message by using XPath extensions

The XPath implementation in WebSphere Message Broker provides the following extra functions for modifying the message tree:

set-local-name(*object***)**
: Sets the local part of the expanded name of the context node to the value specified in the argument. *object* can be any valid expression, and is converted to a string as if a call to the string function is used.

set-namespace-uri(*object***)**
: Sets the namespace URI part of the expanded name of the context node to the value specified in the argument. *object* can be any valid expression, and is converted to a string as if a call to the string function is used.

set-value(*object***)**
: This function sets the string value of the context node to the value specified in the argument. *object* can be any valid expression, and is converted to a string as if a call to the string function is used.

To allow for syntax element trees to be built as well as modified, the following axis is available in addition to the 13 that are defined in the XPath 1.0 specification:

select-or-create::*name* **or ?***name*
: ?*name* is equivalent to select-or-create::*name*. If *name* is @name, an attribute

is created or selected. This selects child nodes matching the specified name, or creates new nodes according to the following rules:

- ?*name* selects children called *name* if they exist. If a child called *name* does not exist, ?*name* creates it as the last child, then selects it.
- ?$*name* creates *name* as the last child, then selects it.
- ?^*name* creates *name* as the first child, then selects it.
- ?>*name* creates *name* as the next sibling, then selects it.
- ?<*name* creates *name* as the previous sibling, then selects it.

Manipulating other parts of the message tree by using a JavaCompute node

You can access parts of the message tree other than the message body data. These parts of the logical tree are independent of the domain in which the message exists, and all these topics apply to messages in all supported domains, including the BLOB domain. You can access all parts of the message tree by using a JavaCompute node.

Elements of the message tree can be accessed in the same way as the message body data, by using a JavaCompute node.

- "Accessing headers by using a JavaCompute node"
- "Updating the local environment with the JavaCompute node" on page 538
- "Updating the Global Environment with the JavaCompute node" on page 538

Accessing headers by using a JavaCompute node:

You can access headers by using a JavaCompute node.

If an input node receives an input message that includes message headers that the input node recognizes, the node starts the correct parser for each header. Parsers are supplied for most WebSphere MQ headers. The following topics provide guidance for accessing the information in the MQMD and MQRFH2 headers that you can follow when accessing other headers that are present in your messages.

- "Copying message headers by using a JavaCompute node"
- "Accessing the MQMD header by using a JavaCompute node" on page 537
- "Accessing the MQRFH2 header by using a JavaCompute node" on page 537

For further details of the contents of these and other WebSphere MQ headers for which WebSphere Message Broker provides a parser, see "Element definitions for message parsers" on page 1517.

Copying message headers by using a JavaCompute node:

You can copy message headers by using a JavaCompute node.

The *Modifying Message Class* template in the JavaCompute node wizard generates the following code to copy message headers using a JavaCompute node:

```
public void copyMessageHeaders(MbMessage inMessage, MbMessage outMessage) throws MbException
{
 MbElement outRoot = outMessage.getRootElement();
 MbElement header = inMessage.getRootElement().getFirstChild();

 while(header != null && header.getNextSibling() != null)
 {
```

```
  outRoot.addAsLastChild(header.copy());
  header = header.getNextSibling();
 }
}
```

Accessing the MQMD header by using a JavaCompute node:

WebSphere MQ and WebSphere MQ Everyplace messages include an MQMD header. You can use a JavaCompute node to refer to the fields in the MQMD, and to update them.

Note that SCADA messages also include an MQMD header.

The following Java code shows how to add an MQMD header to your message:

```
public void addMqmd(MbMessage msg) throws MbException
{
 MbElement root = msg.getRootElement();

 // create a top level 'parser' element with parser class name
 MbElement mqmd = root.createElementAsFirstChild("MQHMD");

 // specify next parser in chain
 mqmd.createElementAsFirstChild(MbElement.TYPE_NAME_VALUE,
  "Format",
  "XMLNS");
}
```

Accessing the MQRFH2 header by using a JavaCompute node:

You can use a JavaCompute node to add an MQRFH2 header to an outgoing message.

When you construct MQRFH2 headers in a JavaCompute node, two types of field exist:

- Fields in the MQRFH2 header structure (for example, Format and NameValueCCSID)
- Fields in the MQRFH2 NameValue buffer (for example mcd and psc)

The following code adds an MQRFH2 header to an outgoing message that is to be used to make a subscription request:

```
public void addRfh2(MbMessage msg) throws MbException
{
 MbElement root = msg.getRootElement();
 MbElement body = root.getLastChild();

 // insert new header before the message body
 MbElement rfh2 = body.createElementBefore("MQHRF2");

 rfh2.createElementAsFirstChild(MbElement.TYPE_NAME_VALUE, "Version", new Integer(2));
 rfh2.createElementAsFirstChild(MbElement.TYPE_NAME_VALUE, "Format", "MQSTR");
 rfh2.createElementAsFirstChild(MbElement.TYPE_NAME_VALUE, "NameValueCCSID", new Integer(1208));

 MbElement psc = rfh2.createElementAsFirstChild(MbElement.TYPE_NAME, "psc", null);
 psc.createElementAsFirstChild(MbElement.TYPE_NAME, "Command", "RegSub");
 psc.createElementAsFirstChild(MbElement.TYPE_NAME, "Topic", "department");
 psc.createElementAsFirstChild(MbElement.TYPE_NAME, "QMgrName", "QM1");
 psc.createElementAsFirstChild(MbElement.TYPE_NAME, "QName", "PUBOUT");
 psc.createElementAsFirstChild(MbElement.TYPE_NAME, "RegOpt", "PersAsPub");
```

```
 MbXPath xp = new MbXPath("/MQMD/Format" + "[set-value(´MQHRF2´)]", root);
 root.evaluateXPath(xp);
}
```

In this example, the MQHRF2 parameter is the parser class name, which is different from the parser element name (MQRFH2). For a list of the parsers, root element names, and class names for different headers, see Available parsers.

Updating the local environment with the JavaCompute node:

The local environment tree is part of the logical message tree in which you can store information while the message flow processes the message.

The following information shows how to update the local environment:

1. Make a new copy of the local environment to update it. Use the full version of the copy constructor to create a new MbMessageAssembly object, as shown in the following example:

```
MbMessage env = assembly.getLocalEnvironment();
MbMessage newEnv = new MbMessage(env);

newEnv.getRootElement().createElementAsFirstChild(
  MbElement.TYPE_NAME_VALUE,
  "Status",
  "Success");

MbMessageAssembly outAssembly = new MbMessageAssembly(
  assembly,
  newEnv,
  assembly.getExceptionList(),
  assembly.getMessage());

getOutputTerminal("out").propagate(outAssembly);
```

2. Edit the copy to update the local environment.

Updating the Global Environment with the JavaCompute node:

The Global Environment tree is always created when the logical tree is created for an input message. However, the message flow neither populates it nor uses its contents. You can use this tree for your own purposes, for example to pass information from one node to another. You can use the whole tree as a scratchpad or working area.

The Global Environment can be altered across the message flow, therefore do not make a copy of it to alter. The following Java code shows how to alter the Global Environment:

```
MbMessage env = assembly.getGlobalEnvironment();
env.getRootElement().createElementAsFirstChild(MbElement.TYPE_NAME_VALUE, "Status", "Success");

getOutputTerminal("out").propagate(assembly);
```

Accessing the ExceptionList tree using Java:

The ExceptionList tree is created with the logical tree when an input message is parsed.

The tree is initially empty, and is only populated if an exception occurs during message flow processing. It is possible that more than one exception can occur; if more than one exception occurs, the ExceptionList tree contains a subtree for each exception.

You can access the ExceptionList tree in JavaCompute nodes from the `MbMessageAssembly` parameter of your `evaluate()` method.

You can access the ExceptionList tree in a node in an error handling procedure. For example, you might want to route the message to a different path based on the type of exception.

You can use the querying capabilities within XPath to carry out this task. The descendant axis (//) of XPath gives you the ability to search for an element by name regardless of its depth in the tree. For example:

```
//ParserException
```

returns all elements in the tree named `ParserException`.

If you are specifically interested in a particular message, you can use a predicate (see predicates for further information) to narrow down the result set. For example:

```
//ParserException[Number=5016]
```

returns only the exception that contains `Number=5016`.

If you only want to extract the text message associated with this exception, the following XPath expression returns this as a `java.lang.String`:

```
string(//ParserException[Number=5016]/Text)
```

The following Java code extracts this text from your code:

```
String text =
(String)inAssembly.getExceptionList().evaluateXPath("string(//ParserException[Number=5016]/Text)");
```

For information on accessing the ExceptionList tree using ESQL, see “Accessing the ExceptionList tree using ESQL” on page 357

Accessing broker properties from the JavaCompute node

For each broker, WebSphere Message Broker maintains a set of properties. You can access some of these properties from your Java programs. It can be useful, during the run time of your code, to have real-time access to details of a specific node, flow, or broker.

Broker properties are divided into four categories:

- Properties that relate to a specific node
- Properties that relate to nodes in general
- Properties that relate to a message flow
- Properties that relate to the execution group

Broker properties that are accessible from ESQL and Java includes a table that shows the groups of properties that are accessible from Java. The table also indicates if the properties are accessible from ESQL.

Broker properties have the following characteristics.

- They are grouped by broker, execution group, flow, and node.

- They are case sensitive. Their names always start with an uppercase letter.
- They return NULL if they do not contain a value.

To access broker properties in a JavaCompute node, call methods on the following classes:

- MbBroker
- MbExecutionGroup
- MbMessageFlow
- MbNode

For example:

```
String brokerName = getBroker().getName();
```

Accessing message flow user-defined properties from a JavaCompute node

Customize a JavaCompute node to access properties that you have associated with the message flow in which the node is included.

To access these properties from a JavaCompute node, use the `getUserDefinedAttribute(`*name*`)` method, where *name* is the name of the property that you are accessing. The type of the object that is returned depends on the type of the property that you are accessing. The object has one of a set of types:

- MbDate
- MbTime
- MbTimestamp
- Boolean
- byte[]
- String
- Integer 32-bit values
- Long 64-bit values
- Double
- BigDecimal
- BitSet

You cannot access user-defined properties in the constructor. To access them at initialization time, implement the following method, and use it to access the user-defined properties.

```
public void onInitialize() throws MbException
{
   // access the user-defined properties here
}
```

You can use the Configuration Manager Proxy (CMP) to change the value of user-defined properties. Use the getUserDefinedPropertyNames(), getUserDefinedProperty(), and setUserDefinedProperty() methods to query, discover, and set user-defined properties, as described in Setting user-defined properties at run time in a CMP application.

Adding keywords to JAR files

If a BAR file contains JAR files, you can associate keywords with the JAR files.

Before you start:

Read about keywords in "Message flow version and keywords" on page 66.

To add keywords to a JAR file:

1. Add a file called `META-INF/keywords.txt` to the root of the JAR file.
2. Add your keywords to the `META-INF/keywords.txt` file, because this file is parsed for keywords when it is deployed. Keywords have this format:

   ```
   $MQSI keyword = value MQSI$
   ```

For example, a deployed BAR file contains *compute.jar*, and *compute.jar* contains the file `META-INF/keywords.txt` with the following contents:

```
# META-INF/keywords.txt
$MQSI modified date = 3 Nov MQSI$
$MQSI author = john MQSI$
```

This content means that the keywords "modified date" and "author" are associated with the deployed file *compute.jar* in Configuration Manager Proxy applications, including the Message Broker Toolkit.

You have now added keywords to your JAR file.

Next:

When you have added keywords to your JAR file, you can display them in the BAR file editor; for more information, see "Version and keyword information for deployable objects" on page 265.

Interacting with databases by using the JavaCompute node

Access databases from Java code included in the JavaCompute node.

Choose from the following options for database interaction:

- Broker JDBCProvider for type 4 connections
- MbSQLStatement
- JDBC API in an unmanaged environment
- SQLJ

If you use JDBCProvider for type 4 connections or MbSQLStatement, the databases that you access can participate in globally coordinated transactions. In all other cases, database access cannot be globally coordinated.

Broker JDBCProvider for type 4 connections:

You can establish JDBC type 4 connections to interact with databases from your JavaCompute nodes. The broker supports type 4 drivers, but does not supply them. You must obtain these drivers from your database vendor; for information about supported drivers, see Supported databases.

Use the broker JDBCProvider for type 4 connections to benefit from the following advantages:

- Use broker configuration facilities to define the connection, and to provide optional security, in preference to coding these actions.
- Configure the broker and the databases to coordinate access and updates with other resources that you access from your message flows, except when the broker is running on z/OS.
- Use the broker Java API getJDBCType4Connection to initiate the connection, then perform SQL operations by using the standard JDBC APIs. The broker

manages the connections, thread affinity, connection pooling, and life cycle. If a connection is idle for approximately 1 minute, or if the message flow completes, the broker closes the connection.

If the broker is running on a distributed system, you can configure the databases and the connections to be coordinated with other resource activity. Global coordination on distributed systems is provided by WebSphere MQ, and can include interactions with local or remote databases, including remote databases that are defined on z/OS systems. If you establish a JDBC type 4 connection to a database from a broker that is running on z/OS, coordination is not provided. For information about setting up connections and coordination, see Enabling JDBC connections to the databases.

Before you can include this function in the code that you write for the node, you must configure the required environment. Decide whether your database requires security of access, and whether you want the database updates to participate in globally coordinated transactions. For the required and optional tasks, see Enabling JDBC connections to the databases.

When you have configured the JDBCProvider, you can establish a JDBC type 4 connection to the database by using the getJDBCType4Connection call on the MbNode interface. The following code provides an example of its use:

```
public class MyJavaCompute extends MbJavaComputeNode {
  {
    public void evaluate(MbMessageAssembly inAssembly) throws MbException {
      MbOutputTerminal out = getOutputTerminal("out");
      MbMessage inMessage = inAssembly.getMessage();

      // create new message
      MbMessage outMessage = new MbMessage(inMessage);
      MbMessageAssembly outAssembly = new MbMessageAssembly(inAssembly,outMessage);

      try {
        // Obtain a java.sql.Connection using a JDBC Type4 datasource - in this example for a
        // JDBC broker configurable service called "MyDB2"

        Connection conn = getJDBCType4Connection("MyDB2",
                     JDBC_TransactionType.MB_TRANSACTION_AUTO);

        // Example of using the Connection to create a java.sql.Statement
        Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                     ResultSet.CONCUR_READ_ONLY);
        ResultSet srs0 = stmt.executeQuery("SELECT NAME, CITY FROM MySchema.MyTable");

        stmt.executeUpdate("UPDATE MySchema.MyTable SET CITY = \"Springfield\" WHERE Name = \"Bart\"");
        .
        // Perform other database updates
        .

      } catch (SQLException sqx ){
        sqx.printStackTrace();
      } finally {
        // clear the outMessage
        outMessage.clearMessage();
      }

    }
  }
}
```

In this example:

- *MyDB2* is the name of the JDBCProvider configurable service. Use the name of the service that you have created to connect to your database.
- *MySchema* is the name of the database schema (not the name of the database).
- MB_TRANSACTION_AUTO defines the level of transaction coordination that is required by the node. Only this value is supported, and indicates that the coordination in the node is inherited from the coordination configured at message flow level.

Because the broker is managing the connections, your code must comply with the following restrictions:

- The only valid location for the getJDBCType4Connection call is in the JavaCompute node's evaluate method. Do not code the getJDBCType4Connection call in a JavaCompute node constructor in case the connection is lost and needs to be reestablished. SQL connections are cached by the broker and might be released because of inactivity, or if a communications failure with the database is detected. Therefore, make the getJDBCType4Connection call in the evaluate method, which returns the existing cached connection if it is available, or a new connection if it is not.
- Do not include code that performs a COMMIT or a ROLLBACK function.
- Do not close the connection to the database.

To indicate a failure (and roll back a transaction), issue an exception from the JavaCompute node and the broker will handle the rollback.

MbSQLStatement:

The MbSQLStatement class provides full transactional database access by using ESQL and ODBC. The broker resource manager coordinates database access when using MbSQLStatement. Global coordination is provided by WebSphere MQ on distributed systems, and by RRS on z/OS. For information about how to set up the ODBC resources that are required, see Enabling ODBC connections to the databases.

Create instances of the MbSQLStatement class by using the createSQLStatement() method of MbNode, passing to the method the ODBC data source, a broker EQSL statement, and, optionally, the transaction mode.

- Calling select() on this object returns the results of the query.
- Calling execute() on this object executes a query where no results are returned, such as updating a table.

The following Java code shows how to access a database by using MbSQLStatement:

```
MbMessage newMsg = new MbMessage(assembly.getMessage());
MbMessageAssembly newAssembly = new MbMessageAssembly(assembly, newMsg);

String table = "dbTable";

MbSQLStatement state = createSQLStatement( "dbName",
 "SET OutputRoot.XMLNS.integer[] = PASSTHRU('SELECT * FROM " + table + "');" );

state.setThrowExceptionOnDatabaseError(false);
state.setTreatWarningsAsErrors(true);
state.select( assembly, newAssembly );

int sqlCode = state.getSQLCode();
if(sqlCode != 0)
```

```
{
 // Do error handling here
}

getOutputTerminal("out").propagate(assembly);
```

JDBC API in an unmanaged environment:

You can access standard Java APIs in the code that you write for your JavaCompute nodes, including JDBC calls. You can therefore use JDBC APIs to connect to a database, write to or read from the database, and disconnect from the database. The broker allows your JDBC connection code to invoke both type 2 and type 4 JDBC drivers in this environment, but does not supply them. You must obtain these drivers from your database vendor.

If you choose this method to access databases, the broker does not provide support for managing the transactions; your code must manage the local commit and rollback of database changes. Your code must also manage the connection life cycle, connection thread affinity, and connection pooling. You must also monitor the access to databases when you use this technique to ensure that these connections do not cause interference with connections made by the broker. In particular, be aware that type 2 drivers bridge to an ODBC connection that might be in use in message flows that access databases from ESQL.

SQLJ:

SQLJ is a Java extension that you can use to embed static SQL statements within Java code. Create SQLJ files by using the workbench. The broker resource manager does not coordinate database access when using SQLJ.

1. Enable SQLJ capability in the workbench:
   a. Switch to the Broker Application Development perspective.
   b. Select **Window** → **Preferences**.
   c. Expand **General**.
   d. Select **Capabilities**.
   e. Select **Data**.
   f. Click **OK**.
2. Create a SQLJ file within a Java project:
   a. Right-click the ***Java project*** in which you want to create the file.
   b. Select **New** → **Other**.
   c. Expand **Data**.
   d. Expand **SQLJ Applications**.
   e. Select **SQLJ File**.
   f. Click **Next**.
   g. Follow the directions given by the New SQLJ File wizard to generate the SQLJ file.

You can now reference the class in this SQLJ file from a JavaCompute node class in this project or in another referenced project.

Calling an Enterprise Java Bean

You can call an Enterprise Java Bean (EJB) from a JavaCompute node.

Before you start:

- Ensure that all required Java classes are in WebSphere Message Broker's shared-classes directory, or are referenced in the CLASSPATH environment variable. You can use the wildcard character (*) at the end of a directory path specifier to load all JARs in that directory path.
- Ensure that the user JAR files that are needed for EJB access are referenced in CLASSPATH. For more information, see the documentation for the application server that is hosting the EJB.
- If you are using a version of WebSphere Message Broker before Version 6.0 Fix Pack 3, you must set the context loader by including the following statement in the node's Java code before the InitialContext is set:

  ```
  Thread currentThread().setContextClassLoader(this.getClass().getClassLoader());
  ```

The following example shows how to call an EJB from a JavaCompute node:

```
public class CallAckNoAckEJB_JavaCompute extends MbJavaComputeNode {

   public void evaluate(MbMessageAssembly inAssembly) throws MbException {
                MbOutputTerminal out = getOutputTerminal("out");
                MbOutputTerminal alt = getOutputTerminal("alternate");

                MbMessage inMessage = inAssembly.getMessage();

                // create new message
                MbMessage outMessage = new MbMessage(inMessage);
                MbMessageAssembly outAssembly = new MbMessageAssembly(inAssembly,outMessage);

                try {
                        // ----------------------------------------------------------
                        // Add user code below

                        String response = null;
                        String responseMessage = null;

                        Properties properties = new Properties();
                        properties.put(Context.PROVIDER_URL, "iiop://localhost:2809");
                        properties.put(Context.INITIAL_CONTEXT_FACTORY, "com.ibm.websphere.naming.
WsnInitialContextFactory");

                        try {

                           Context initialContext = new InitialContext(properties);
                                Object obj = initialContext.lookup("ejb/com/acme/ejbs/AckNoAckHome");
                                AckNoAckHome ejbHome = (AckNoAckHome)javax.rmi.PortableRemoteObject.
narrow(obj,AckNoAckHome.class);

                                AckNoAck ackNoAck = ejbHome.create();
                                responseMessage = ackNoAck.getAck();
                                response = "Ack";
                        } catch(Exception e) {
                                responseMessage = e.getMessage();
                                response = "NoAck";
                        }

                        MbElement cursor = outMessage.getRootElement().getFirstElementByPath("/XML/AckNoAck");
                        cursor.createElementAsLastChild(MbElement.TYPE_NAME,"Response",null);
                        cursor.getLastChild().createElementAsLastChild(MbElement.TYPE_NAME,response,null);
                        cursor.getLastChild().getLastChild().createElementAsLastChild(MbElement.TYPE_VALUE,null,
responseMessage);

                        // End of user code
                        // ----------------------------------------------------------

                        // The following should only be changed
                        // if not propagating message to the 'out' terminal
```

```
                out.propagate(outAssembly);

            } finally {
                // clear the outMessage
                outMessage.clearMessage();
            }
    }
}
```

JavaCompute node Exception handling and the Failure terminal

You do not need to catch exceptions that are thrown in a JavaCompute node. The broker handles exceptions automatically.

If you catch an exception in your code, throw it again, allowing the broker to construct an exception list and propagate the message to the failure terminal, if one is connected. If you have not connected the failure terminal, the exception is thrown back to a Catch node or an input node.

Logging errors with the JavaCompute node

The MbService class contains a number of static methods for writing to the Event log or syslog. Define message catalogs by using Java resource bundles to store the message text.

Three levels of severity are supported:
- Information
- Warning
- Error

The following sample demonstrates the use of resource bundles and logging:
- JavaCompute Node

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

Developing message mappings

Message mappings define the blueprint for creating a message. The topics in this section describe message mappings and explain how to develop them.

Concept topics:
- “Message mappings overview” on page 547
- “Message flows, ESQL, and mappings” on page 9
- “Advanced schema structures” on page 549

Task topics:
- “Creating message mappings” on page 551
- “Creating a message map file in the Broker Development view” on page 551
- “Creating a message map file from a Mapping node” on page 552
- “Configuring message mappings” on page 553
- “Mapping from source: by selection” on page 555
- “Mapping from source: by name” on page 555
- “Mapping a target element from source message elements” on page 564
- “Setting the value of a target element to a constant” on page 566
- “Setting the value of a target element to a WebSphere MQ constant” on page 566

A further section of topics contain reference information about message mapping:

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

Message mappings overview

Message mappings define the blueprint for creating a message, where the created message is known as the target message.

Messages can contain the following components:

- Simple elements and attributes
- Complex elements (structures)
- Repeating simple or complex elements
- Other (embedded) messages

Messages can contain protocol-specific headers, which might need to be manipulated by WebSphere Message Broker. Dynamic setting of a message destination (routing) within the WebSphere Message Broker might also be required.

Values for target message elements can be derived from:

- Input message elements (the input message is also known as the source message)
- Database tables
- Constant values
- WebSphere MQ constants
- Functions supplied by the Mapping node
- User-defined functions

The logic to derive values can be simple or complex; conditional statements might be needed, as might loops, summations, and other functions. All of the mappings shown earlier can be achieved using a Mapping node.

You can also create a reusable form of message map known as a submap. Submaps can be called from message maps and ESQL code.

You must have message definitions for any messages that you want to include in a message mapping. You can select the messages from your existing message definitions when you create a new message map. The Mapping node supports the following message domains:

MRM
XMLNSC
XMLNS
MIME
SOAP
DataObject
JMSMap
JMSStream
XML
BLOB
IDOC

If you use an unsupported parser to perform mappings, for example a user-defined parser, error messages might be generated when messages pass through your message flow. See "Changing the target message domain" on page 601 for information about setting the message domain for the target message.

Find out more about message flows, ESQL, and mappings.

This section also contains information about "Advanced schema structures" on page 549.

Message flows, ESQL, and mappings

A message flow represents the set of actions that are performed on a message when it is received and processed by a broker.

The content and behavior of a message flow is defined by a set of files that you create when you complete your definition and configuration of the message flow content and structure:

- The message flow definition file `<message_flow_name>.msgflow`. This file is mandatory, and is created automatically for you. It contains details about the message flow characteristics and contents (for example, what nodes it includes, its promoted properties, and so on).
- The ESQL resources file `<message_flow_name>.esql`. This file is required only if your message flow includes one or more of the nodes that you can customize by using ESQL modules. You can create this file yourself, or you can cause it to be created for you by requesting specific actions against a node.

  You can customize the following built-in nodes by creating free-form ESQL statements that use the built-in ESQL statements and functions, and your own user-defined functions:
  - Compute
  - Database

– Filter

- The message mappings file `<message_flow_name><_nodename>.msgmap`. This file is required only if your message flow contains one or more of the nodes that you can customize by using mappings. You can create this file yourself, or you can cause it to be created for you by requesting specific actions against a node. A different file is required for each node in the message flow that uses the Message Mapping editor.

  You can customize the following built-in nodes by specifying how input values map to output values.

Node	Usage
"DataDelete node" on page 923	Use this node to delete one or more rows from a database table without creating an output message.
"DataInsert node" on page 926	Use this node to insert one or more rows in a database table without creating an output message.
"DataUpdate node" on page 930	Use this node to update one or more rows in a database table without creating an output message.
"Mapping node" on page 1052	Use this node to construct output messages and populate them with information that is new, modified from the input message, or taken from a database. You can also use the Mapping node to update, insert, or delete rows in a database table.
"Warehouse node" on page 1307	Use this node to store all or part of a message in a database table without creating an output message.

You can use built-in ESQL functions and statements to define message mappings, and you can use your own ESQL functions.

The "Extract node" on page 944 also uses mappings to create a new output message that contains a subset of the contents of the input message. However, the Extract node is deprecated in WebSphere Message Broker Version 6.0 and later releases. Although message flows that contain an Extract node remain valid in WebSphere Message Broker Version 6.0, redesign your message flows to replace Extract nodes by Mapping nodes to take advantage of later enhancements.

Advanced schema structures

You can use several advanced schema structures in mappings.

This section contains information about the following subjects:

- "Substitution groups"
- "Wildcards" on page 550
- "Derived types" on page 550
- "List types" on page 550
- "Union types" on page 550

Substitution groups: A substitution group is an XML Schema feature that provides a means of substituting one element for another in an XML message. The element that can be substituted is called the *head* element, and the substitution group is the list of elements that can be used in its place.

The head element and any mapped substitutions are shown by default in the Source and Target panes of the Message Mapping editor. The mapped substitutions are listed beneath the head element. You can show and hide the substituting elements displayed in the Source and Target panes by selecting **Show Substituting Elements**. You create mappings to or from members of substitution groups in the same way as you would map other elements.

An abstract head element of a substitution group is not displayed and when substitution is blocked, the substitution group folder is not displayed.

Wildcards: A mapping that you perform to or from a wildcard results in a submap call. Specify the wildcard replacement when you choose the parameter of a submap call.

A wildcard element or attribute can be instantiated only with another element or attribute. The Message Mapping editor allows only a global element or attribute as a wildcard replacement.

Derived types: For an element of a given type, the base type and the mapped derived types are shown by default in the Source and Target panes of the Message Mapping editor. All attributes and elements of the base and derived types are listed under each type respectively. You can show and hide the derived types displayed in the Source and Target panes by selecting **Show Derived Types**.

You create mappings to or from a derived type and its contents in the same way that you would map any type or type content. When you map a derived type element, the Message Mapping editor generates ESQL code with the appropriate xsi:type attribute.

List types: A list type is a way of rendering a repeating simple value. The notation is more compact than the notation for a repeating element and provides a way to have multi-valued attributes.

You map list type attributes or elements in the same way that you would map any other simple type attribute or element. Mapping between two list type elements is the same as mapping between any two simple type elements.

To transform between a list type and a non-list type, such as a repeating element, write an ESQL function, then package the function as a map. The Message Mapping editor automatically selects this submap as the default transformation for the list type.

Union types: A union type is the same as a union of two or more other simple types and it allows a value to conform to any one of several different simple types.

Use the Message Mapping editor to create mappings to or from union type attributes or elements in the same way as you would map atomic simple type attributes or elements, as shown in the following diagram:

```
<xsd:simpleType name="zipUnion">
 <xsd:union memberTypes="USState listOfMyIntType"/>
</xsd:simpleType>
<xsd:element name=zip type=zipUnion/>
```

Creating message mappings

The topics in this section describe how to create message mappings. Most actions can be achieved either by using the menu bar, or by right-clicking and choosing an action from a drop-down menu. For consistency, the following topics describe the menu bar method:

Creating a message map file in the Broker Development view

You can create a message map file for use in your message flows in the Broker Development view. If you want to add a database to your message map file, you must have created a database definition for the database.

To create a message map (.msgmap) file in the Broker Development view:

1. From the Broker Application Development perspective, click **File** → **New** → **Message Map**.

   Alternatively, in the Broker Development view, right-click the message flow project that you want to create the message map in and click **New** → **Message Map**.

   The New Message Map wizard opens.
2. Specify the Project, Name and Schema for the message map. The project list is filtered to show only projects in the active working set. Click **Next**.
3. Follow the on-screen instructions to complete the New Message Map wizard:
   a. On the **Select map kind and its source and target** pane, select the type of map you want to create.

- If you select the option **Message map called by a message flow node**, a message map is created that can be accessed from a node. Properties are associated with any source or target messages, and you can select to include message headers and the LocalEnvironment with the message body.
- If you select the option **Submap called by another map**, a message map is created that can be referenced from another message map. This is known as a submap and can contain components of a message body such as global elements, global attributes, and global types. A submap does not contain Properties, message headers, or the LocalEnvironment.

b. Expand the Messages, Message Components, and Data Sources folders and their children, to display all the available options. Select the Messages, Message Components, Database Selects, Stored Procedures, and User Defined Functions that you want to use for your map from **Select Map Sources**. The stored procedure list, if any, shows the stored procedure signature, such as the procedure name, its parameter mode (IN/OUT/INOUT), parameter name, and the data type.

c. Expand the Messages and Data Targets folders and their children, to display all the available options. Select the Messages, Table Inserts, Table Updates and Table Deletes that you want to use as targets for your map from **Select Map Targets**.

The information contained inside a square bracket is the data source name, schema name, and the database project which contains the database definition.

Messages, data sources and data targets are filtered to show only resources from the active working set. If you cannot find the Messages, Message Components, Data Sources or Data Targets that you expect, clear the **Apply working set filtering to artifact selection(s) on the page** check box.

4. Select **Finish** to create the new message map. The "Message Mapping editor" on page 1530 opens with the selected sources and targets.

After you have created a message map file, configure the message mappings. You must also configure the **Mapping routine** property on your mapping node to match the name of your new mapping file.

Creating a message map file from a Mapping node

You can use a Mapping node to create a message map with messages and databases as both sources and targets.

Before creating a message map file, ensure you complete the following tasks:

1. "Creating a message flow project" on page 256
2. "Creating a message flow" on page 259
3. Define message flow content that includes a Mapping node, see "Defining message flow content" on page 268.

To create a message map (`.msgmap`) file from a Mapping node:

1. Open your message flow from the Broker Application Development perspective.
2. Double-click the Mapping node, or right-click the Mapping node and click **Open Map**. The New Message Map for Mapping Node wizard opens.
3. Select the combination of Messages, Data Sources, or both, that you want to use as sources for your map from Select map sources. Select the combination of Messages, Data Targets, or both, that you want to use as targets for your map from Select map targets.

If you cannot find the Messages, Data Sources or Data Targets that you expect, clear the **Apply working set filtering to artifact selection(s) on the page** check box.

4. Click **OK** to create the new message map. The Message Mapping editor opens with the selected sources and targets, for more information see "Message Mapping editor" on page 1530.

After you have created a message map file, you can configure the message mappings, see "Configuring message mappings."

Configuring message mappings

Use the Message Mapping editor to configure a message mapping.

The editor provides the ability to set values for:

- Message destination
- Message content
- Message headers

See the "Mapping node" on page 1052 topic for more information about how to set the properties of a Mapping node.

Wizards and dialog boxes are provided for tasks such as adding mappable elements and working with submaps. Mappings that are created with the Message Mapping editor are validated and compiled automatically, ready to be added to a broker archive (BAR) file, and for subsequent deployment to WebSphere Message Broker.

Use the Message Mapping editor to perform the following tasks:

Common tasks:

- "Mapping a target element from source message elements" on page 564
- "Mapping a target element from database tables" on page 583
- "Mapping a target element from database stored procedures" on page 585
- "Mapping a target element from database user-defined functions" on page 586
- "Setting the value of a target element to a constant" on page 566
- "Setting the value of a target element using an expression or function" on page 568
- "Configuring conditional mappings" on page 570
- "Configuring mappings for repeating elements" on page 571

Message destination tasks: You might map a destination so that the destination can be set dynamically, by setting values in LocalEnvironment.Destination. You can also retrieve information after a message has been sent, by accessing information in LocalEnvironment.WrittenDestination.

- "Configuring the LocalEnvironment" on page 573
- "Mapping headers and folders" on page 573

Message content tasks:

- "Adding messages or message components to the source or target" on page 575
- "Adding a database as a source or target" on page 575
- "Showing or hiding substituting elements in the Message Mapping editor" on page 554

- "Showing or hiding derived types in the Message Mapping editor"

Message header tasks:

- "Configuring message headers" on page 573
- "Mapping headers and folders" on page 573

Showing or hiding substituting elements in the Message Mapping editor:

You can use the **Show Substituting Elements** dialog box to show and hide substituting elements in the Message Mapping editor.

The head element and any mapped substitutions are shown by default in the Message Mapping editor. You can use the **Show Substituting Elements** dialog to show and hide the substituting elements in the Source and Target panes of the Message Mapping editor. To show or hide substituting elements:

1. In either the Source or the Target pane, right-click the element that you want to show or hide the substituted elements for. If an element has substituted elements, it is displayed as a substitutions folder in the Source or the Target pane.
2. Click **Show Substituting Elements** on the pop-up menu. The Show Substituting Elements dialog box is displayed.
3. In the Show Substituting Elements dialog box, select elements to show them in the Message Mapping editor, or clear them to hide them in the Message Mapping editor. You cannot hide any element that is already used in a mapping.
4. Click **OK**.

The elements that you have chosen to show or to hide are stored as preferences in your workspace.

Showing or hiding derived types in the Message Mapping editor:

You can use the **Show Derived Types** dialog box to show and hide derived types in the Message Mapping editor.

For an element of a given type, only the base type and any mapped derived types are shown in the Message Mapping editor. You can use the **Show Derived Types** dialog to show and hide derived types in the Source and Target panes of the Message Mapping editor. To show or hide derived types in the Message Mapping editor:

1. In either the Source or the Target pane, right-click the element that you want to show or hide the derived types for. If an element has derived types it is displayed as a specializations folder in the Source or the Target pane.
2. Click **Show Derived Types** on the pop-up menu. The Show Derived Types dialog box is displayed.
3. In the Show Derived Types dialog box, select elements to show them in the Message Mapping editor, or clear them to hide them in the Message Mapping editor. You cannot hide any element that is already used in a mapping.
4. Click **OK**.

The elements that you have chosen to show or to hide are stored as preferences in your workspace.

Mapping from source: by selection

You can map from source fields by using Map from Source, or by using the drag-and-drop method.

Using Map from Source

1. Select the source and target elements that you want to map by clicking them. (Ctrl+click to select multiple source or target elements.)
2. Click **Map → Map from Source**.

   There are four possible scenarios that result in mapping by selection using Map from Source.

   - If more than one mappable source element is selected, the selected sources are mapped to the selected target.
   - If more than one mappable target element is selected, the selected source is mapped to the selected targets.
   - If one mappable source and one mappable target are selected, and neither element has any children, the selected source is mapped to the selected target.
   - If one mappable source and one mappable target are selected, where both the elements have children and the same type definition, the selected source is mapped to the selected target.

Using the drag-and-drop method

Drag the appropriate source element or elements onto the target element or elements (Ctrl+click to select multiple source or target elements.)

When you use the drag-and-drop method to map from source, mapping by selection is always performed. You can use the drag-and-drop method in the following scenarios:

- More than one mappable source element is selected. In this case, the selected sources are mapped to the selected target.
- More than one mappable target element is selected. In this case, the selected source is mapped to the selected targets.
- One mappable source and one mappable target are selected, and neither element has any children. In this case, the selected source is mapped to the selected target.
- The selected source and target elements have the same type definition, or when the source type is derived from the target type. In this case the entire structure below the element is copied.

In other scenarios, when a mapping by selection is not possible, the Map by Name wizard opens to enable a Map by Name mapping to be performed instead.

Mapping from source: by name

The Map by Name wizard is used to map complex types by examining the names of source elements and target elements to create mappings.

The Map by Name wizard can also be used to map database columns. The following steps describe how to map from source using the Map by Name wizard.

Using the Map by Name wizard

1. Click the source and target complex elements, database, schema, table, database stored procedure, or database user-defined function that you want to map.

2. Click **Map → Map by Name**. The Map by Name wizard opens to allow you to perform mapping by name.
3. Choose the appropriate option from the Select how to map from source to target wizard:
   - **Map leaves of the selected nodes**. This option maps the leaves of the source element to the leaves of the target element, that match each other; this option is selected by default.
   - **Map immediate children of the selected nodes**. This option maps only the immediate children of the source element to the immediate children of the target element that match each other. This option is available only when the selected source and target elements have immediate children that are mappable.
4. After selecting the **Map leaves of the selected nodes** or **Map immediate children of the selected nodes** option, specify how names are matched.
   - **Case sensitive**. This option enables you to select whether you want to match the case sensitivity of the name; this option is not selected by default.
   - **Alphanumeric characters (Letters and digits only)**. This option excludes special characters (for example & and -) from the name; this option is selected by default.

   The preceding two options are independent of one another, and you can select their values separately.
5. Specify the Mapping Criteria between the sources and targets.

   If the source and target names that you are using satisfy more than one of the following options, the order in which names are matched is:

   a. Same
   b. Synonym
   c. Similar

   Any target that is matched during an earlier step is excluded from name matching in a later step.

   - **Create mappings between sources and targets with the same name**. This option matches items of the same name, and is selected by default.

     Whether the two names are considered to be the same, depends on your selections for **Case sensitive** and **Alphanumeric characters (Letters and digits only)**.

     For example, if you have used the default options for **Case sensitive** and **Alphanumeric characters (Letters and digits only)**, GIVEN_NAME and GivenName are considered to be a match.

     However, if you have selected **Case sensitive** as well as **Alphanumeric characters (Letters and digits only)**, the two names are considered to be identical only if they contain the same alphanumeric characters in the same order, and the characters are of the same case.

     See “Mapping by Same Name” on page 558 for further information.
   - **Create mappings when source and target names are more similar than**. This option allows you to specify how similar two words have to be to create a mapping between them by varying the result from zero to 100 percent. The result is displayed and the default value is 60; see “Sample similarity values” on page 557 for some examples of how similar words are matched to one another.
   - **Create mappings between source and target names defined as synonyms in file**. This option allows you to create mappings for word pairs that are

defined in a synonym file. A synonym file is a flat text file with file extension `.txt` or `.csv`. See "Creating and using a synonym file" on page 562 for further information about how you create a synonym file from a spreadsheet written in Microsoft® Excel.

See "Format of the synonym file" on page 559 for further information about the synonym file itself, and "Algorithm used to match synonyms" on page 563 for further information about the methods that are used to match synonyms in a synonym file.

6. Click **Finish** to complete the process, or **Next** to obtain further options.

The Map by Name wizard opens automatically when you use the drag-and-drop method to map from source where the source and target are complex types with different type definitions, or where the source type is not derived from the target type.

Selecting matches:

Use the Map by Name wizard to select the mappings that you want to create.

When you have specified how you require the names to be matched on the initial panel of the Map by Name wizard, and have selected **Next**, you obtain a panel that displays all the matches found.

You can now select the options that you require:

1. Select a row in the **Selectable Mapping Targets** column that you want to change. Selecting a folder tree node results in the entire tree branch being selected or not selected.
2. Click **Edit** to start the **Select Mapping Source** dialog.
3. To select a mapping target, select the appropriate tree node check box. Conversely, to remove a mapping target, clear the appropriate tree node check box.
4. Ensure that you have selected only the number of matches that you require. The third column displays the number of sources selected for each mapping target. The cell has a value greater than one when the source of a map contains several elements of certain names under various containers, and the source names match to the same target name.
5. Click **Finish** to complete the mapping process, or click **Back** to change the matches that you have set up. When you click **Finish**, you obtain a warning message if either, or both, of the following conditions apply:
   a. More than a few sources to map to the same target.
   b. Many targets for which you want to create mappings.

Sample similarity values:

The following table lists words that are similar to one another, together with their similarity value in percent.

Word1	Word2	Similarity value %
catalog	catalogue	85
anasthesia	anaesthesia	84
recognize	recognise	75
color	colour	66

Word1	Word2	Similarity value %
theater	theatre	66
tire	tyre	33
intro	introduction	53
abbr	abbreviation	42
name	fullname	60
firstname	fullname	40
id	identification	14
NCName	Non colonized name	40
USA	United States of America	0
faq	frequently asked questions	0

Mapping by Same Name:

Options available when you select **Create mappings between sources and targets with the same name** or a similar name.

When you select **Create mappings between sources and targets with the same name**, the following rules apply:

1. Any target field that has a fixed value is excluded in name matching. Any target that is already mapped, or under a container that is already mapped, is excluded from name matching.
2. If a source and a target have the same name, it is a match, regardless of the kind of, and XSD type of, the source and target. An element, an attribute, and a database column can all form a match if their names are the same.
3. XML namespaces are excluded from name matching. Therefore, `abc:something` and `xyz:something` are considered the same, as are `{http://www.abc.com}:something` and `{http://www.xyz.com}:something`.
4. When multiple sources have the `same` name as one target, one mapping is created. However, when multiple targets have the `same` name as one source, multiple mappings are created, each for one source and one target.
5. When performing `Map from Source` for a selected source and a selected target, the workbench might insert some `for` and `if` statements based on the repeatability (maximum occurrences) of the selected source and target, and their containers.

   The same process occurs for `Map by Name`, based on the repeatability of the selected sources and targets, and their containers.

   However, there are not any `for` or `if` statements inserted on descendants of the selected source and target.
6. When you select the **Map leaves of the selected nodes** option, the following steps are taken to match names:
   a. Compare the path name starting after the selected source or target.
   b. Compare the item name without path

   For example, if you invoke the action **Create mappings between sources and targets with the same name** and have a:

   - Source path for `partNum` of `$source/po:purchaseOrder/`**`items`**`/item/partNum`, where **items** is the selection that you made in the source.

- Target path for `partNum` of `$target/`**`po:purchaseOrder`**`/items/item/partNum`, where **po:purchaseOrder** is the selection that you made in the target.

During step a) the source and target path names involved in the same-name test are `item/partNum` and `/items/item/partNum`.

During step b) the source and target item names that are involved in same-name test are `partNum` and `partNum`; that is, name matches are done using short names without their paths.

Note that sources and targets matched in a previous step do not participate in a later step.

Mapping by similar name

1. Fixed value targets and mapped targets are excluded in name matching; see Point 1 in the preceding section.
2. The *similarity* test is done using the name of an element, an attribute, or a database column regardless of its type; see Point 2 in the preceding section.
3. The *similarity* test applies in the same way to case sensitivity and alphanumeric characters as for **Mapping by same name**.
4. Namespace or namespace prefixes do not participate in the *similarity* test; see Point 3 in the preceding section.
5. The behavior for the situation when multiple sources are similar to one target, and when multiple targets are similar to one source, is the same as Point 4 in the preceding section.
6. The repeatability (maximum occurrences) of containers and descendants of the selected source and target are handled in the same way as in Point 5 in the preceding section.
7. When you select the option **Create mappings when source and target names are more similar than**, you must also select **Create mappings between sources and targets with the same name**.
8. When you select **Map leaves of the selected nodes**, the following steps are taken to match names.

   Sources and targets matched in a previous step do not participate in a later step:

   - The path names starting after the selected source and target are the same.
   - The item names excluding the path are the same.
   - The item names excluding the path are similar.
9. You can select the similarity threshold for two words to be considered similar.
10. You cannot use any other similarity algorithm.

Format of the synonym file:

`Map by Name` allows you to create mappings between specific sources and targets by putting the names of the sources and targets in a file called the synonym file.

Synonyms, in the context of the synonym file, are groups of words that represent mappings that you want to create.

File type:

A synonym file can reside anywhere in your file system, only if the encoding used in the synonym file is the same as that used by the Eclipse Toolkit system.

However, if the synonym file uses a specific encoding that is, or might be, different from the encoding of the Eclipse Toolkit, the file must reside in a project in the workbench.

If the synonym file is created outside the workbench, and uses a specific encoding, save the file under a workbench project and click **Refresh** to make the file visible in the navigator.

The synonym file uses Tab-separated or comma-separated files only. If you have written your mapping requirement in any external application, for example, Microsoft Word or Microsoft Excel, you must export the relevant data in a format that the synonym file supports.

Item names in the file:

A synonym file contains the names of items to be mapped, without the path to the item or the namespace of the item.

For example, if you want to map `partNum` to `partNumber` in the following XML, you must put `partNum` in the synonym file, not `item/partNum`, `items/item/partNum`, or `purchaseOrder/items/item/partNum`.

```
<po:purchaseOrder xmlns:po="http://www.ibm.com">
 <items>
    <item>
         <partnum>100-abc</partnum>
         <productName>Acme Integrator</productName>
         <quantity>22</quantity>
         <USPrice>100.99</USPrice>
         <po:comment>Acme Integrator</po:comment>
         <shipDate>2008-12-01</shipDate>
    </item>
 </items>
</po:purchaseOrder>
```

Synonyms in the file can:

- Be case sensitive or not case sensitive
- Contain the entire mapping item name
- Have non alphanumeric characters removed

You must select the appropriate options in the action dialog, according to the way in which you created the synonym file.

Rows in the synonym file:

In the synonym file, each row represents one group of names to be mapped between each other and each row must contain at least two names. Names within a row are separated by commas in `.csv` files, and by Tab characters in `.txt` files.

A synonym file can contain an optional special row at the top. This top row contains key words **Source**, **Target**, or **Source_Target**, separated by the same delimiter used in the remainder of the file. The top row is used to indicate whether the synonyms are to be used to match names in mapping the source or the target:

- If the first word in the top row is **Target**, the first name only, in each subsequent row is searched in the mapping target for name matching.
- If the second word in the top row is **Source**, the second name only, in each subsequent row is searched in the mapping source for name matching.

- If the third word in the top row is **Source_Target**, the third name only, in each subsequent row is searched in both the mapping source and mapping target for name matching.

The top row must not contain fewer key words than the maximum number of names in any row in the file.

If the top row contains any word other than **Source**, **Target**, or **Source_Target**, the top row is ignored and it is assumed that the top row is missing. If you omit the optional top row, every name in the synonym file is considered to be **Source_Target**; that is, any name found either in the mapping source or in the mapping target is matched.

If a synonym file contains two rows:

car	*automobile*
automobile	*vehicle*

car and *vehicle* are not considered to be synonyms.

In order to make all three words synonyms, your synonym file can have either:

- One row with all three words -

car	*automobile*	*vehicle*

or

- Three rows -

car	*automobile*
automobile	*vehicle*
car	*vehicle*

Special characters:

You can write synonym files manually, or export them from another application; for example, Microsoft Excel. Item names in synonym files reflect the application domain and do not have to match exactly the names in the XML schema or the relational database column.

For example, a synonym file might contain the row:

```
summer      l'été
```

As `l'été` does not conform to the XML NCName format, you could name the element `l_été`. As long as all the alphanumeric characters in the synonym file match those in the schema, you can use the file with the option **Letters and digits only, ignore non-alphanumeric characters**.

Many mapping requirements are written in Microsoft Excel, and cells in a Microsoft Excel file might contain specific characters like double quotation marks, space, new line, comma, and so on. When such a Microsoft Excel file is saved as a Tab-separated or comma-separated file, they contain additional double quotation marks.

Two groups of synonyms in a synonym file are delimited either by a Line Feed (LF) character, or Line Feed followed by a Carriage Return (LFCR). A Carriage Return (CR) character by itself does not end a group of synonyms.

Leading and trailing space characters adjacent to the delimiter (comma or Tab character) are ignored. Blank rows, or rows that contain only space characters, are permitted and ignored in a synonym file.

Different editors might inject different space characters into a synonym file; spaces are not used to delimit synonyms, and spaces are ignored unless they are inside double quotation marks.

If a synonym contains a comma, a double quotation mark, a carriage return, or a leading or trailing space that is significant, the synonym must be enclosed in double quotation marks. A double quotation mark within a synonym is ended by another double quotation mark. For example:

```
"comma,separated"
"double""quote"
  "with<CR>
   newline"
  "  spaces   "
```

When the synonym file is read by the workbench, the double quotation marks at the beginning and end of the synonym are removed and the workbench stores the following in the synonym table:

```
comma,separated
double"quote
  with<CR>newline
    spaces
```

The workbench reads a synonym file containing these special characters correctly, and you should select the **Letters and digits only, ignore non-alphanumeric characters** option when using the synonym file.

Creating and using a synonym file:

You can create a synonym file manually, or by generating a synonym file from the information that is contained in a Microsoft Excel spreadsheet, using the following set of instructions.

The following set of instructions describes how to create a synonym file where the original mapping requirement is written in Microsoft Excel. If your original requirement is written in a table in Microsoft Word you must copy and paste the table into Microsoft Excel before you begin.

Amend the process to allow for extra facilities that your enterprise uses.

1. Select the section of the Microsoft Excel spreadsheet that you require. For example, if you have a Product that you want to map to a Part number, select that section of the spreadsheet.
2. Remove all columns from the spreadsheet, except the ones containing the source field name and the target field name.

   You might have to edit some of the cells. For example, if your mapping instruction includes the phrase `based on`, remove this phrase.
3. If the source or target fields contain paths, remove the paths to leave only the short names of the item.

However, it is helpful to sort the column before removing the paths. Sorted path names can indicate which is the best source or target to select when invoking the action.

If all the interested sources (or targets) start with the same path prefix, you might consider selecting the lowest source (or target) node in the tree which has that common path prefix.

4. Remove all rows that do not have a source field name and a target field name. For example, if you have an obsolete product that no longer has a part number and you have put `n/a` in the source, remove that row.
5. Select the **Save As** function in Microsoft Excel to save the spreadsheet into a format acceptable by our workbench.

   You can use either a Tab delimited `.txt` file or a comma delimited `.csv` file.

   A comma delimited file can be opened using Microsoft Excel and it looks like the original Microsoft Excel file; the file can also be viewed using a text editor.
6. Create the mappings using the synonym file.

   Select the options in the **Map by Name** wizard that match your requirements; for example, select the default options of **Map leaves of the selected nodes** and **Alphanumeric characters (Letters and digits only)**.

   When you have chosen these options, select **Create mappings between source and target names defined as synonyms in file**.

   If you need to map same-name sources to targets, and the synonym file does not contain rows with those names (for example a row with **car,car**), you might want to check the **Create mappings between sources and targets with the same name** option, in addition to **Create mappings between source and target names defined as synonyms in file** option.

   For that matter, you can even check both **Create mappings between sources and targets with the same name** and **Create mappings when source and target names are more similar than**, in addition to **Create mappings between source and target names defined as synonyms in file**, if your synonym file does not contain a row **color,colour** and you want to map between them.
7. Click **Finish**.
8. Edit mapping expressions based on the requirement that you need.

   For example, if you have the following value:

   ```
   PRODUCT  $SOURCE/Batch/Detail/Replace/PartNumber
   ```

   edit the mapping expression to:

   ```
   xs:boolean($source/Batch/Detail/Replace/PartNumber = 1)
   ```
9. Create all the mappings that the Map by Synonym process has not generated. Use the drag-and-drop method, **Map from Source**, or the **Enter Expression** process.

Algorithm used to match synonyms:

The way in which synonyms are matched with `Map by Name` to create mappings between specific sources and targets follows a particular set of rules.

1. Fixed value targets and mapped targets are excluded in name matching; see point 1 in “Mapping by Same Name” on page 558.
2. The synonym matching is done by using the name of an element, an attribute, or a database column regardless of its type; see point 2 in “Mapping by Same Name” on page 558.

3. The synonym matching of alphanumeric characters is not case-sensitive, and is identical to that used in "Mapping from source: by name" on page 555.
4. Namespace or namespace prefixes do not participate in synonym matching; see point 3 in "Mapping by Same Name" on page 558.
5. The behavior for the situation when multiple sources are synonyms of one target, and when multiple targets are synonyms of one source, is the same as point 4 in "Mapping by Same Name" on page 558.
6. The repeatability (maximum occurrences) of containers and descendants of the selected source and target are handled in the same way as in point 5 in "Mapping by Same Name" on page 558.
7. If a source and a target have the `same` name, they are not considered a match under the option for synonyms. If you require a mapping for same-name sources and targets, you must also select the **same name** option.
8. In addition to mapping synonyms, you might want to create mappings for some, but not all, same-name sources and targets. In this case, you have two options:
   - Clear **Create mappings between sources and targets with the same name**, and include the same-name sources and targets in the synonym file
   - Select **Create mappings between sources and targets with the same name**, and clear the unwanted mappings on the second page of the wizard.
9. When you select **Map leaves of the selected nodes** together with both same name and synonym mapping options, the following steps are taken to match names.

   Sources and targets matched in a previous step do not participate in a later step:
   - The path names starting after the selected the source and target are the same.
   - The item names excluding the path are the same.
   - The item names excluding the path are synonyms
10. When you select **Map leaves of the selected nodes**, and you require synonyms to be mapped without mapping same names, the item names only are checked for synonyms; paths are ignored.

Mapping a target element from source message elements

You can map:

- simple source elements to simple target elements
- source structures to target structures (where the source and target are of the same type)
- source structures to target structures (where the source and target are of a different type)
- multiple simple source elements to a simple target element

The following sections describe how to perform mapping for these particular scenarios using the Message Mapping editor.

Mapping simple source elements to simple target elements

In the following example, the source element called Name does not contain the same children as the target element called Name:

Source	Target
Name Title First_name Middle_name Last_name	Name Title First_names Last_name

To map one of the child elements, drag the element from the Source pane onto the corresponding element in the Target pane; for example, drag the Last_name source element onto the Last_name target element.

The mapping is represented by a line between the source element and the target element and an entry for the mapping in Xpath format appears in the Spreadsheet pane. A triangular icon indicates which elements in the Source and Target panes have been mapped.

Mapping source structures to target structures (where the source and target are of the same type)

In the following example, the source element called Name has the same structure as the target element called Name:

Source	Target
Name Title First_name Middle_name Last_name	Name Title First_name Middle_name Last_name

To map the entire source structure to the target structure, drag the parent element (Name) from the Source pane onto the corresponding element (Name) in the Target pane. All the child elements are mapped.

Mapping source structures to target structures (where the source and target are of a different type)

In the following example, the source element called Name has a different structure to the target element called DifferentName:

Source	Target
Name Title First_name Middle_name Last_name	DifferentName Title FirstName LastName

To map the entire source structure to the target structure, drag the parent element (Name) from the Source pane onto the corresponding element (DifferentName) in the Target pane. The Map By Name wizards opens. Select **Map leaves** and **Map items of same and similar names** to map all child elements in the target. The source element Middle_name will not be mapped, as there is no target element with the same or a similar name.

Mapping multiple source elements to a simple target element

In the following example, you want to concatenate the First_name and Middle_name source elements to form a single target element called First_names:

Source	Target
Name Title First_name Middle_name Last_name	Name Title First_names Last_name

To map multiple source elements to a simple target element, Ctrl+click the appropriate source elements (First_name and Middle_name) and the target element (First_names), then click **Map → Map from Source**. A concatenate function appears in the Spreadsheet pane; you can edit this function to define how the concatenated target element looks, for example, by adding a white space between the two source elements.

To customize the target element (for example, to make the target value equal to the source value plus one), see "Setting the value of a target element using an expression or function" on page 568. You cannot map a simple element if one of its ancestors also has a mapping. For example, you cannot map Properties from source to target, then map Properties/MessageFormat.

Setting the value of a target element to a constant

Use the Message Mapping editor to set the value of a target element to a constant.

1. In the Target pane, right-click the target element or attribute and click **Enter Expression**. If the target element or attribute has a default value, this value is added to the Edit pane.
2. Enter the required constant in the Edit pane and click **Enter**. When entering the constant, observe the following rules:
   - Enclose string element values in single quotation marks.
   - Enter numeric element values without quotation marks.
   - For boolean element values enter 0 for false or 1 for true, without quotation marks. Alternatively, you can enter the fn:false() function for false, or the fn:true() function for true.

   The Spreadsheet pane is updated with the value that you have defined.

You cannot set a value for a simple element if one of its ancestors also has a mapping. For example, you cannot map Properties from source to target, then set a value for Properties/MessageFormat.

You can also set a target element to a WebSphere MQ constant or an ESQL constant.

Setting the value of a target element to a WebSphere MQ constant

There are two ways to set the value of a target element to a WebSphere MQ constant, depending on whether the target element has an entry in the Map Script column of the Message Mapping editor Spreadsheet pane.

- If the target element has an entry in the Map Script column:
  1. In the Spreadsheet pane, select the target element.
  2. Enter `$mq:` followed by the WebSphere MQ constant in the Edit pane.
  3. Press **Enter**.

  The Spreadsheet pane is updated with the expression for a WebSphere MQ constant.
- If the target element does not have an entry in the Map Script column:

1. In the Target pane, right-click the target element and click **Enter Expression**.
2. Enter `$mq:` followed by the WebSphere MQ constant in the Edit pane.
3. Press **Enter**.

   The Spreadsheet pane is updated with the expression for a WebSphere MQ constant.

The following examples demonstrate how to enter a WebSphere MQ constant in the Edit pane:

```
$mq:MQ_MSG_HEADER_LENGTH
$mq:MQMD_CURRENT_VERSION
```

When the map is saved a warning message is displayed if the expression entered for the WebSphere MQ constant is incorrect, for example, if the constant is not recognized. This is an example of the warning message: `The target "$target/purchaseOrder/comment" is not referencing a valid variable`.

Content Assist (**Edit** → **Content Assist**) provides a list of the WebSphere MQ constants available.

1. Select `$mq: ( MQ constants )`
2. Select **Edit** → **Content Assist** again to display a list of the available constants.

WebSphere MQ constants that can be used as values for target elements, grouped by the parameter or field to which they relate, can be found in the *WebSphere MQ Constants* book.

Setting the value of a target element to an ESQL constant

There are two ways to set the value of a target element to an ESQL constant, depending on whether the target element has an entry in the Map Script column of the Message Mapping editor Spreadsheet pane.

- If the target element has an entry in the Map Script column:
  1. In the Spreadsheet pane, select the target element.
  2. Enter `$esql:` followed by the ESQL constant in the Edit pane.
  3. Press **Enter**.

     The Spreadsheet pane is updated with the expression for a WebSphere MQ constant.
- If the target element does not have an entry in the Map Script column:
  1. In the Target pane, right-click the target element and click **Enter Expression**.
  2. Enter `$esql:` followed by the ESQL constant in the Edit pane.
  3. Press **Enter**.

     The Spreadsheet pane is updated with the expression for a WebSphere MQ constant.

The following examples demonstrate how to enter an ESQL constant in the Edit pane:

```
$esql:ValidateLocalError
$esql:ParseComplete
```

When the map is saved a warning message is displayed if the expression entered for the ESQL constant is incorrect, for example, if the constant is not recognized. This is an example of the warning message: `The target "$target/purchaseOrder/comment" is not referencing a valid variable`.

Content Assist (**Edit → Content Assist**) provides a list of the WebSphere MQ constants available.

1. Select `$esql: ( ESQL Constants )`
2. Select **Edit → Content Assist** again to display a list of the available constants.

Setting the value of a target element using an expression or function

There are two ways to set the value of a target element to an expression, depending on whether the target element has an entry in the Map Script column of the Message Mapping editor Spreadsheet pane:

- If the target element has an entry in the Map Script column:
  1. In the Spreadsheet pane, select the target element.
  2. Enter the required expression in the Edit pane.
  3. Press **Enter**.

  The Spreadsheet pane is updated with the value or expression.
- If the target element does not have an entry in the Map Script column:
  1. In the Target pane, right-click the target element and click **Enter Expression**. If the target element has a default value, this value is added to the Edit pane.
  2. Enter the required expression in the Edit pane.
  3. Press **Enter**.

  The Spreadsheet pane is updated with the value or expression.

The following examples demonstrate techniques for entering mapping expressions in the Edit pane.

- If the target element is derived from a source element, drag the source element or elements onto the Edit pane; for example:

  ```
  $source/Properties/MessageSet
  ```
- Use arithmetic expressions, such as:

  ```
  $source/Properties/Priority + 1
  ```
- Use mapping, Xpath or ESQL function names. Content Assist (**Edit → Content Assist**) provides a list of available functions. For example:

  ```
  esql:upper($source/Properties/ReplyIdentifier)
  ```
- You can perform casting in the Edit pane; for example:

  ```
  xs:string($source/Properties/CodedCharSetId)
  ```

You cannot enter an expression for a simple element if one of its ancestors also has a mapping. For example, you cannot map Properties from source to target, then set a value of Properties/MessageFormat.

Creating a BLOB output message using a message map

Use the Message Mapping editor to create a bit stream from a message source, and create it as a BLOB output message.

Before you start:

Create a mapping that includes a BLOB message as a target; see "Creating a message map file from a Mapping node" on page 552.

Take the following steps:

1. Right-click the BLOB message that you want to map in the Target pane, and select **Enter Expression** from the menu.
2. In the Edit pane, type `esql:asbitstream()`.
3. Drag the source field to the Edit pane, placing it between the parentheses, for example:

   ```
   esql:asbitstream($source/po:purchaseOrder)
   ```

   Alternatively, you can use content assist to select the `esql:asbitstream` function. In the Edit pane press Ctrl+Space to display a list of available functions and associated parameters. The `asbitstream` function is an ESQL Field function. The function can take other parameters; see "Predefined ESQL mapping functions" on page 1544.

   When you move the cursor out of the Edit pane, or press Enter, the mapping is displayed between the fields in the Source and Target panes.

Mapping from a BLOB message to an output message

Use the Message Mapping editor to parse a BLOB message.

Before you start:

Create a mapping; see "Creating a message map file from a Mapping node" on page 552.

Take the following steps:

1. Right-click the element in the target pane, and select **Enter Expression** from the menu.
2. In the Edit pane, type `msgmap:element-from-bitstream()`.
3. Drag the BLOB to the Edit pane, placing it between the parentheses, for example:

   ```
   msgmap:element-from-bitstream($source/BLOB)
   ```

   Alternatively, you can use content assist to select the `msgmap:element-from-bitstream` function. In the Edit pane press Ctrl+Space to display a list of available functions and associated parameters. The function can take other parameters; see "Predefined mapping functions" on page 1551. When you move the cursor out of the Edit pane, or press Enter, the mapping is displayed between the fields in the Source and Target panes.

Deleting a source or target element

The following steps describe how to delete source and target elements using the "Message Mapping editor" on page 1530:

- To delete a source path, modify the expression so that it no longer uses the source value to compute the target.

  If this is the last use of the source path, the line linking the source and target is removed. If the expression no longer has any value, the target becomes unmapped.
- To delete a target from the Edit pane, click the target and click **Delete**.

  The target structure is preserved if possible.
  - If you delete a "for" row, clicking **Delete** removes the single row.
  - If you have an `if`, an `elseif`, or an `else` statement, clicking **Delete**:
    1. On an `elseif` statement deletes the `elseif` statement and the contents of the statement.

2. On an `else` statement deletes the `else` statement and the contents of the statement.
3. On an `if` statement with a subsequent `elseif` statement, deletes the `if` statement and the contents of the statement, and causes the subsequent `elseif` statement to become an `if` statement.
4. On an `if` statement with only a subsequent `else` statement, deletes everything except the contents of the `else` statement.
5. On an `if` statement with no subsequent statements, deletes everything except the contents of the `if` statement.

- To delete a database source, click the SELECT statement then remove all references to the source manually. Alternatively, delete the SELECT source in the Source pane then remove all references to the source manually.
- To delete a database target, delete the INSERT, UPDATE or DELETE statement. Alternatively, update or delete the statement in the Target pane.

Configuring conditional mappings

How to set the value of a target element conditionally in a Mapping node.

1. In the Spreadsheet pane of the Message Mapping editor, select the target element and click **Map → If**.

   One row is added to the Spreadsheet pane, above the target element:

   - In ths row, Map Script is set to "if".

     Its value is an expression that is evaluated to see whether it is true. If true, the target element is set to the value specified in its "Value" column. Initially, its "Value" column is set to ‘fn:true()’, which means that the condition is always met, and the target element is always set to the "Value" column.

2. Change the expression in the if row’s "Value" column by selecting the cell, or the if row, in the Spreadsheet pane, and setting the value in the Edit pane.

   Amend the expression in the Edit pane to specify the correct condition statement by performing the following steps:

   a. Select any database columns that are pertinent to the condition statement, and drag them from the Source pane into the Edit pane.
   b. Select any source message elements with values that are pertinent to the condition statement, and drag them from the Source pane into the Edit pane.
   c. Open Content Assist by clicking **Edit → Content Assist** and select the functions to be applied to the condition.

3. Add further condition statements by selecting the if row in the Spreadsheet pane, and clicking **Map → Else If**.

   Two rows are added to Spreadsheet pane, below the target element:

   - In the first row, Map Script is set to `elseif`. Process this as described in Step 2.
   - In the second row, Map Script is set to the target element. Its `Value` cell is initially blank. Set this value as described in “Setting the value of a target element to a constant” on page 566, and “Setting the value of a target element using an expression or function” on page 568.

4. To set the value of a target element when the `if` statement is not true, select the `if` statement for the target element in the Spreadsheet pane, and click **Map → Else**.

   Two rows are added to Spreadsheet pane, below the target element:

   - In the first row, Map Script is set to `else`. You cannot enter anything in the Value column of this row.

- In the second row, Map Script is set to the target element; its value is initially blank. Set this value as described in "Setting the value of a target element to a constant" on page 566, and "Setting the value of a target element using an expression or function" on page 568.

Configuring mappings for repeating elements

To configure the Mapping node to process repeating elements, use the 'For' option in the Message Mapping editor Spreadsheet pane. The following combinations of repeating elements are possible:

- repeating source and non-repeating target
- non-repeating source and repeating target
- repeating source and repeating target

By default, if the source is a database, it is processed as a repeating source.

Configuring a repeating source and a non-repeating target:

To map a repeating source element to a non-repeating target element, drag elements between the Message Mapping editor Source and Target panes.

The following items appear in the Spreadsheet pane:

- A "for" row with Value set to the repeating source element.
- An "if" row with Value set to `msgmap:occurrence($source/...) = 1`.
- A row with Map Script set to the target field and Value set to the source field.

The first occurrence of the source field is mapped to the target field. The "for"' row specifies that a loop is to be iterated for the specified repeating element. The if row restricts the logic to a single occurrence of the repeating element. See "Configuring conditional mappings" on page 570 for more information on conditional logic in a mapping node.

1. To map an occurrence other than the first, change the expression in the if row to `msgmap:occurrence($source/...) = n`, where *n* is the occurrence that you want to map.

   If the repeating source field is within one or more repeating structures, a hierarchy of `for` and `if` rows is placed in the Spreadsheet pane, one for each level of repetition.
2. If the source field contains a numeric data type, mapping all occurrences of a repeating source field to a non-repeating target results in the sum of all the source elements. Perform this mapping by selecting the source element and target element and clicking **Map → Accumulate**.

   This action sets the following value in the Spreadsheet pane for the target element:

   ```
   fn:sum($source/...)
   ```

   The result of the accumulate action is a numeric value. If your target has a different data type, you must cast the result to the appropriate type for the selected target. For example, if your target is xs:string type, you must alter the results of the accumulate action from `fn:sum($source/x/y/z)` to `xs:string(fn:sum($source/x/y/z))`, in order to cast the result to the correct data type for your target.

   You cannot map different occurrences of a repeating source element to different non-repeating target elements.

Configuring a non-repeating source and a repeating target:

To map a non-repeating source element to a repeating target element, drag elements between the Message Mapping editor Source and Target panes.

The first occurrence of the target element is set to the value of the source element.

To map to an occurrence other than the first, complete the following steps:

1. If the target element is not shown in the Spreadsheet pane, right-click its lowest ancestor row, then click **Insert Children**. Repeat this action until the target element is shown.
2. Right-click the target element and click **Insert Sibling After** or **Insert Sibling Before** to select the location to insert the repeating target elements. The **Insert Sibling After** or **Insert Sibling Before** options are not enabled if there is nothing valid to be inserted at the selected location. Selecting either of these opens the Insert Sibling Statement wizard.
3. Select the element to insert from the list of valid items.
4. Enter the number of instances to be added and click **OK**. The number of instances to be added must be less than or equal to the maximum occurrence specified for the selected element.

The specified number of instances of the repeating target element are added to the Spreadsheet pane. The inserted statements do not have a mapping expression and any children are not displayed. Right-click each element, then click **Insert Children** to display any child elements.

By repeating the **Insert Sibling After** and **Insert Sibling Before** action, it is possible to insert more repeating elements in the target than the maximum occurrence specifies. Verify that the number of repeating elements is valid, and delete any unwanted entries.

Configuring a repeating source and a repeating target:

To map a repeating source element to a repeating target element drag elements between the Message Mapping editor Source and Target panes. The following items appear in the Spreadsheet pane:

- A 'for' row with Value set to the repeating source element.
- A row with Map Script set to the target field and Value set to the source field.

All occurrences of the source element are mapped to the respective occurrences of the target element. You can map repeating source structures to repeating target structures if the source and target are of the same complex type.

Populating a message map

Use the Insert Children wizard to add elements from the Target pane to the Spreadsheet pane. The Insert Children wizard creates child structures for the selected parent structure.

When you add a message target to a message map, $target in the Spreadsheet pane is populated by default with Properties and the message body root. The Properties fields MessageSet, MessageType and MessageFormat, are added together with their default values, unless the selected message is in the BLOB domain. Other message elements and their children can be added to the Spreadsheet pane without creating

mappings by using the Insert Children wizard. The following steps show how to populate the Spreadsheet pane with other message elements using the Insert Children wizard:

Using the Insert Children wizard

1. Right-click a parent element in the Spreadsheet pane and click **Insert Children**. The Insert Children wizard is displayed.
2. Select the items you wish to create mappings for. Items required in the target message are selected by default. The selected items are added to the Spreadsheet pane.
3. Repeat **Insert Children** to add further child elements to the Spreadsheet pane.

If any target elements are missing warning messages are displayed in the Message Mapping Editor. These warning messages indicate the name and expected position of the missing elements. You can used the Insert Children wizard to add the missing elements.

You can also use the Insert Children wizard to add target elements to the Spreadsheet pane when there are existing mappings. Any existing mappings are not altered by the wizard.

If the target map is a submap the Spreadsheet pane is populated by default with the selected element or attribute root. You can use the Insert Children wizard in the submap to add any child elements to the Spreadsheet pane in the same way.

Configuring the LocalEnvironment

You can set values in the LocalEnvironment in the same way as setting values in other elements of a message. Add the LocalEnvironment to your message map using the **Add or Remove Headers and Folders** dialog as described in "Mapping headers and folders." If you set any values in the target LocalEnvironment, set the mapping mode property for the Mapping node to a value that contains LocalEnvironment. To do this, select the mapping node in your message flow and click **Properties** → **Basic** → **Mapping Mode**.

You cannot map Local Environment objects that are not listed.

Configuring message headers

You can set values for headers in the same way as setting values in other elements of a message.

Add the appropriate headers to your message map using the **Add or Remove Headers and Folders** dialog box as described in "Mapping headers and folders." If you set any values in the target LocalEnvironment, set the Mapping mode property for the Mapping node to a value that contains LocalEnvironment. To do this, select the mapping node in your message flow and click **Properties** → **Basic** → **Mapping Mode**.

You cannot map headers that are not listed.

Mapping headers and folders

Include message headers and folders for source and target messages in a message map.

Before mapping headers and folders, ensure that you do the following tasks:

1. Create a message flow project
2. Create a message flow
3. Define message flow content
4. Create a message map file from the navigator or create a message map from a node.

The following types of message headers and folders can be included for source and target messages in a message map (note that a submap does not include message headers):

- LocalEnvironment
- Properties
- MQ Headers
- HTTP Headers
- JMS Transport Header
- Email Headers

If you choose not to map message headers or the LocalEnvironment explicitly in your message map, the output message is produced with the same message headers as the input message. When you Populate the message map, the Properties folder for the source and target are displayed in the message map, with MessageSet and MessageType initially set based on the target message.

MessageFormat is set to the default wire format of the message set if the parser domain is MRM. The other properties are blank initially, and the message headers are copied from the input message.

Alternatively, if you choose to map any message headers or the LocalEnvironment in your message map, no message headers are copied from the input message. You must add mappings for these headers to ensure that the target message contains appropriate headers to make a valid output message.

If your target message contains an MQRFH2 header, you must select from either the MQRFH2 or MQRFH2C parser in the Add or Remove Headers and Folders dialog. For more information about the MQRFH2 and MQRFH2C parsers, see "The MQRFH2 and MQRFH2C parsers" on page 1527.

To add message headers or other folders to a message map:

1. Right-click your message map in the Broker Development view and select **Open** or right-click your mapping node and select **Open Map** to open the Message Mapping editor.
2. Right-click `$source` in the Source pane and select **Add or Remove Headers and Folders** to add message headers or other folders to the source message. The Add or Remove Headers and Folders dialog box opens.
3. Ensure that **Selected headers and other folders** is selected. If **No folders (map body element only)** is selected your map is a submap, and cannot have headers associated with it. You can change the submap to a message map by selecting **Selected headers and other folders**.
4. Select the headers that you want to map from the list. If you want to map MQ Headers or HTTP Headers, you must select individual headers by expanding the list. If you are using MQ Headers you must include the MQMD, and so this is automatically selected for you.
5. Click **OK** to add the selected message headers or folders to the message map.

6. Right-click `$target` in the Target pane and select **Add or Remove Headers and Folders** to add message headers or other folders to the output message.
7. Repeat steps 3 to 5 to add the headers and folders that you require to the target message.
8. Configure the message header and folder mappings in the same way as other mappings.

You can use **Add or Remove Headers and Folders** to remove message headers or the LocalEnvironment folder. Right-click on either the `$source` or the `$target` to open the Select Message Headers dialog box. Clear the headers or other folders to remove them from the message map. Removing a message header or other folder from the message map removes any associated mappings that you have created. You can remove the Properties folder from the message map, but all built-in parsers require some values in the Properties folder for the output message.

You can map multiple instances of a header by right-clicking on the header in the Message Mapping editor Spreadsheet pane and selecting **Insert Sibling Before** or **Insert Sibling After**. Select the header from the Insert Sibling Statement dialog.

Adding messages or message components to the source or target

You can add additional messages or message components as sources or targets in your message map. To add a message or message component to a source or target:

1. From the Message Mapping editor, click **Map** → **Add Sources and Targets**

   The Add Map Sources and Targets wizard opens.

   Alternatively, right-click in the Source pane and click **Add Sources** or right-click in the Target pane and click **Add Targets**.
2. Select messages or message components from the message sets that are in your Message Broker Toolkit workspace.

   If you cannot find the messages or message components that you expect, clear the **Apply working set filtering to artifact selection(s) on the page** check box.

   If one does not already exist, a project reference is created from your message flow project to the message set project that contains the selected messages or message components.

You can also add sources and targets by dragging the resources from the Broker Development view in the Broker Application Development perspective onto the source or target pane of your message map. Select resources under Messages or Elements and Attributes or Types from your Message Definitions and drag them onto the source or target pane. If you add a message to the message map, Properties are also added. If you add an element, attribute or type to the to the message map a global element for a submap is created. Your message map must use messages, global elements or global types, but not a combination of more than one type.

A Mapping node can have only one source message, but can have several target messages. Therefore, you cannot add a source message if one already exists.

Adding a database as a source or target

Add a database as a source, and database tables as targets, to message maps that support database mappings.

You must create a database definition for your database before you can add it, or the associated tables, to a message map.

You can add database sources and targets in the Broker Application Development perspective in a number of different ways:

- In an existing message map, click **Select Data Source** to add a database as a source.
  1. In the Spreadsheet pane, select the location to add a database table source to your mapping. For example, select `$target`.
  2. Click **Map → Select Data Source**. Alternatively, right-click in the Spreadsheet pane and click **Select Data Source**. The Select Database As Mapping Source wizard opens.
  3. Select your database from the list. If you cannot find the Data Sources or Data Targets that you expect, clear **Apply working set filtering to artifact selection(s) on the page** to view additional options.
  4. When you select a database source during map file creation, the database source is placed in the map script as a sibling (as opposed to a container) of the one or more message targets in the map. When you create a mapping from an item in the database source, the mapper moves the database source in the map script structure to the lowest container of all mappings that use the database source. You can move the database source in the map script by using drag-and-drop or copy-paste actions.
- You can specify the databases and database tables that you want to use in the New Message Map wizard when you create a message map.
  1. Create a message map file by using **File → New → Message Map**, or by right-clicking your mapping node and selecting **Open Map**.
  2. From **Select map sources**, select the Database Sources for your message map.
  3. From **Select map targets**, select the database tables to use as targets in your message map. If you are not creating a message map from a DataDelete, DataInsert, or DataUpdate node, expand the relevant database operation and select from the list of tables. You can select from the following database operations:
     - **Table Inserts**
     - **Table Updates**
     - **Table Deletes**

     If you cannot find the Data Sources or Data Targets that you expect, clear **Apply working set filtering to artifact selection(s) on the page** to view additional options.
  4. When you select a database source during map file creation, the database source is placed in the map script as a sibling (as opposed to a container) of the one or more message targets in the map. When you create a mapping from an item in the database source, the mapper moves the database source in the map script structure to the lowest container of all mappings that use the database source. You can move the database source in the map script by using drag-and-drop or copy-paste actions.
- In an existing message map, click **Add Sources and Targets** to add a database as a source and database tables as a target.
  1. From the Message Mapping editor, click **Map → Add Sources and Targets**. Alternatively, right-click in the Source pane and click **Add Sources**, or right-click in the Target pane and click **Add Targets**.
  2. From **Select map sources**, select the Database Sources for your message map.

3. From Data Targets, in **Select map targets**, select the database tables to use as targets in your message map. If you are not creating a message map from a DataDelete, DataInsert, or DataUpdate node, expand the relevant database operation, and select from the list of tables. You can select from the following database operations:
   - **Table Inserts**
   - **Table Updates**
   - **Table Deletes**

   If you cannot find the Data Sources or Data Targets that you expect, clear **Apply working set filtering to artifact selection(s) on the page** to view additional options.

   You can map to an element within a View, the name of which is annotated with `(read-only view)`.

   You cannot select View in the Add Maps dialog for **Insert**, **Update**, or **Delete**, because View is read only.
4. When you add a database source using the **Add Sources** action, the database source is placed in the map script as a sibling (as opposed to a container) of the one or more message targets in the map. When you create a mapping from an item in the database source, the mapper moves the database source in the map script structure to the lowest container of all mappings that use the database source. You can move the database source in the map script by using drag-and-drop or copy-paste actions.

When you have added the database as a source:

- The Source pane contains a $db:select entry.
- The Spreadsheet pane contains a $db:select entry.

When you have added the database table as a target:

- The Target pane and Spreadsheet pane contain one of the following entries:
  - a $db:insert entry
  - a $db:update entry
  - a $db:delete entry

  You can change the database operation on a selected table by using the **Change Database Operation** dialog.

You cannot add a database as a source or a target to an Extract node.

Modifying databases using message mappings

Create message mappings to read, update, and write to databases.

Select one or more of the following topics to work with databases by using message mappings:

- "Mapping a target element from database stored procedures" on page 585
- "Mapping a target element from database user-defined functions" on page 586
- "Deleting data from a database with a mapping node" on page 587
- "Creating a database to database mapping" on page 588
- "Adding a database as a source or target" on page 575

Adding database definitions to the workbench:

Use the New Database Definition File wizard to add database definitions to the workbench.

You must have a database definition defined in the workbench to create database mappings. You can also use database definitions in other nodes, such as the Compute node, to validate references to database sources and tables. Database definitions are stored in a data design project. You must associate the data design project with all message flow projects that you want to use the database definitions with.

Complete the following steps to add a database definition to the workbench:

1. Switch to the Broker Application Development perspective.
2. Click **File** → **New** → **Database Definition**. The New Database Definition File wizard is displayed.
3. Select an existing data design project, or click **New** to create a new data design project.
4. Select the database type and version that you want to connect to from the **Database and Version** list. Ensure that you select a database from the list that is supported by WebSphere Message Broker; you can use this wizard in a shell-share environment with other Rational products that support other databases or versions.

   For a list of databases supported by the broker, see Supported databases.
5. Click **Next**.
6. Either select to create a new database connection or select a connection to use from the list of existing connections. If you select to use an existing connection, the existing database definition is overwritten.
7. Click **Next**.
8. If you selected to create a new connection:
   a. Optional: You can enter a custom value for the Connection Name if you clear **Use default naming convention**.
   b. Enter values for the Connection to the database, for example, Database name, Host name and Port number.
   c. Enter values for the User ID and Password to connect to the database. Click **Test Connection** to verify the settings you have selected for your database. The default Port number for a DB2 database is 50000. If the connection fails, enter other values such as 50001, 50002 and so on, for the Port number, and test the connection again.
   d. Click **Next**. An error is generated if any of the connection details are wrong. If you specify a Database that already has a database definition in the data design project, click **Yes** in the Confirm file overwrite window to overwrite the existing database definition.
9. Alternatively, if you selected to use an existing connection:

a. Click **Yes** in the Confirm file overwrite window to overwrite the existing database definition.
b. Enter values for the User ID and Password to connect to the database, and click **Next**.

10. Select one or more database schemas from the list and click **Next**.
11. Select the elements that you require on the Database Elements page. You can select any option, in addition to **Tables**, on the Database Elements page.
    a. Select **Views** to see all the database views in the Data Project Explorer
    b. Select **Routines** to add stored procedures and user defined functions to the database definition file.

    If you select other additional options, the database definition files that you create contain more information than the Compute, Database, or Mapping nodes require.
12. Click **Finish**.
13. Add the data design project as a reference to the message flow project:
    a. Right-click the message flow project, and click **Properties**.
    b. Click **Project References**, and select the data design project from the list to add as a referenced project.
    c. Click **OK**.

A new database definition file is added to your data design project. The database definition file name has the following format:`<database>.dbm`. Database definition files are associated with the Data Project Explorer view and the Data Source Explorer view. Tools are available in these views for working with your databases.

Database definition files in the workbench are not automatically updated. If you make a change to your database, you must re-create the database definition files.

Importing large databases to the workbench:

Use the Data Source Explorer view of the workbench to select which options you require when you import large databases.

You must have a database definition defined in the workbench to create database mappings. The following steps tell you how to import specific element definitions into your database definition.

Complete the following steps to import specific database definitions to the workbench:

1. Open the Data perspective of the workbench.
2. In the Data Source Explorer view select **Connections**, and right-click to start the New Connection wizard.
3. To create a connection:
    a. Optional: You can enter a custom value for the Connection Name if you clear **Use default naming convention**.
    b. Enter values for the Connection to the database, for example, Database name, Host name, and Port number.
    c. Enter values for the user ID and password to connect to the database. Click **Test Connection** to verify that the settings you have selected for your database. The default Port number for a DB2 database is 50000. If the connection fails, enter other values such as 50001, 50002, and so on, for the Port number, and test the connection again.

d. Click **Next**. An error is generated if any of the connection details are wrong.

4. On the Filter panel, enable the filter and select the schemas that you require.
5. Click **Finish** to complete the creation of a database connection.
6. In the Data Source Explorer view, expand the folders of the new Database Connection to locate the database elements for which you need definitions.
7. Select the folder that contains the elements you require, right-click, and select **Filter**.
8. On the Filter panel, enable filtering and select the elements that you require.
9. Click **Finish**. The Data Source Explorer view now shows only the elements that you selected.
10. Switch to the Broker Application Development perspective.
11. Click **File** → **New** → **Database Definition**. The New Database Definition File wizard is displayed.
12. Select an existing data design project, or click **New** to create a data design project.
13. Select the database type and version that you want to connect to from the **Database and Version** list. Ensure that you select a database from the list that is supported by WebSphere Message Broker; you can use this wizard in a shell-share environment with other Rational products that support other databases or versions.

    For a list of databases supported by the broker, see Supported databases.
14. Click **Next**.
15. On the Select Connection panel, select **Use an existing connection**. This connection is the one that you created in steps 1 on page 579 to 9 earlier.
16. Click **Next**.
17. Enter values for the user ID and password to connect to the database, and click **Next**.
18. Select one or more database schemas from the list and click **Next**.
19. Select the elements that you require on the Database Elements page. You can select any option, in addition to **Tables**, on the Database Elements page.
    a. Select **Views** to see all the database views in the Data Project Explorer
    b. Select **Routines** to add stored procedures and user-defined functions to the database definition file.
20. Click **Finish**.
21. Add the data design project as a reference to the message flow project:
    a. Right-click the message flow project, and click **Properties**.
    b. Click **Project References**, and select the data design project you created, from the list, to add as a referenced project.
    c. Click **OK**.
22. Carry out the following procedure, to ensure that only the elements you selected appear in a map:
    a. Drag a Mapping node node onto a flow editor.
    b. Double-click the Mapping node node to bring up the map creation dialog.
    c. Follow the prompts to create the map, selecting the data source that references your new database definition as either a source or target.

Database definition files in the workbench are not automatically updated. If you change your database, you must re-create the database definition files.

Creating a message map file from a DataInsert node:

You can use a DataInsert node to create mappings to insert new data into a database from a message, another database or both.

Before creating a message map file, ensure you do the following:

1. Create a message flow project
2. Create a message flow
3. Define message flow content that includes a DataInsert node
4. Create a database definition

To create a message map (.msgmap) file from a DataInsert node:

1. From the Broker Application Development perspective, open your message flow, right-click your DataInsert node, and click **Open Map**. The New Message Map for Data Insert Node wizard opens.
2. Select the combination of Messages, Data Sources or both that you want to use as sources for your map from Select map sources.

   If you cannot find the Messages or Data Sources that you expect, clear the **Apply working set filtering to artifact selection(s) on the page** check box.
3. From the Select map targets pane, select the tables under Table Inserts into which you want to insert new data. The tables that you select are added to the new message map as targets.
4. Select **OK** to create the new message map. The "Message Mapping editor" on page 1530 opens with the selected sources and targets.

After you have created a message map file, you can now configure the message mappings.

Creating a message map file from a DataUpdate node:

You can use a DataUpdate node to create mappings to update existing data in a database from a message, another database or both.

Before creating a message map file, ensure you do the following:

1. Create a message flow project
2. Create a message flow
3. Define message flow content that includes a DataUpdate node
4. Create a database definition

To create a message map (.msgmap) file from a DataUpdate node:

1. From the Broker Application Development perspective, open your message flow, right-click your DataUpdate node, and click **Open Map**. The New Message Map for Data Update Node wizard opens.
2. Select the combination of Messages, Data Sources or both that you want to use as sources for your map from Select map sources.

   If you cannot find the Messages or Data Sources that you expect, select the **Show all resources in workspace** check box.
3. From the Select map targets pane, select the tables under Table Updates in which you want to update data. The tables that you select are added to the new message map as targets.
4. Select **OK** to create the new message map. The "Message Mapping editor" on page 1530 opens with the selected sources and targets.

After you have created a message map file, you can now configure the message mappings.

Creating a message map file from a DataDelete node:

You can use a DataDelete node to create mappings to delete data from a database based on information from an input message, another database or both.

Before creating a message map file, ensure you do the following:

1. Create a message flow project
2. Create a message flow
3. Define message flow content that includes a DataDelete node
4. Create a database definition

To create a message map (.msgmap) file from a DataDelete node:

1. From the Broker Application Development perspective, open your message flow, right-click your DataDelete node, and click **Open Map**. The New Message Map for Data Delete Node wizard opens.
2. Select the combination of Messages, Data Sources or both that you want to use as sources for your map from Select map sources.

   If you cannot find the Messages or Data Sources that you expect, clear the **Apply working set filtering to artifact selection(s) on the page** check box.
3. From the Select map targets pane, select the tables under Table Deletes from which you want to delete data. The tables that you select are added to the new message map as targets.
4. Select **OK** to create the new message map. The "Message Mapping editor" on page 1530 opens with the selected sources and targets.

After you have created a message map file, you can now configure the message mappings.

Change database operation of a message map:

If you have created a message map that performs a database operation such as data insert, data update or data delete on a database table you might want to change the database operation that the map performs.

You might also have created a database mapping by dragging a table from the Broker Development view onto a message map and want to change the default insert operation to another database operation.

To change the database operation of a database table in your message map:

1. From the Broker Application Development perspective, open your message map.
2. Right-click on the target database table in the target pane and click **Change Database Operation**. The Select Database Operation dialog is displayed.
3. Select the database operation you want to perform on the selected table:
   - Insert
   - Update
   - Delete
4. Click **OK** to change the database operation on the selected table.

If you change the database operation of your message map to or from data delete you must re-create any mappings to your target database columns.

Mapping from a message and database:

You can create a message map that uses both a message and a database as a source.

Before creating a message map file, ensure you complete the following steps:

1. Create a message flow project
2. Create a message flow
3. Define message flow content
4. Create database definitions

The following instructions describe how to specify a message and a database as the data source.

1. Right-click a node that supports mapping, such as the Mapping node, and click **Open Map**.
2. Follow the on-screen instructions to complete the New Message Map wizard:
   a. Select the combination of Messages and Data Sources that you want to use as sources for your message map from Select map sources.
   b. Select the combination of Messages, Data Targets or both that you want to use as targets for your map from Select map targets.
3. Perform mapping as usual from the source message.
4. Follow the guidance in "Mapping a target element from database tables" to create the mappings from a source database to the target message or database table.
5. Follow the guidance in "Mapping a target element from database stored procedures" on page 585 to create the mappings from a database stored procedure to the target message or database table.
6. Follow the guidance in "Mapping a target element from database user-defined functions" on page 586 to create the mappings from a database user-defined function to the target message or database table.

Mapping a target element from database tables:

To map a target element from a database table, set up the Mapping node to retrieve the relevant rows from the database and populate the message target elements with values from database.

You can add a database as a source for a mapping in several ways, as described in "Adding a database as a source or target" on page 575. After you have added a database to the mapping, the Spreadsheet pane contains a $db:select entry in the Map Script column. By default, its value is fn:true(), which means that all rows are retrieved from the database table. In database SQL, you can restrict the number of rows by adding a WHERE clause to a database call. In the Mapping node, the equivalent method of restricting the number of selected rows is to use a $db:select expression.

Complete the following steps to restrict the number of rows that are selected in a Mapping node:

1. In the Spreadsheet pane, click the $db:select row. fn:true() is entered in the Edit pane.

2. Edit the expression in the Edit pane to specify the correct condition for the database call. To help you achieve this condition, you can:
   a. Select all database columns that are relevant to the rows that are retrieved, and drag them from the Source pane to the Edit pane. These database column names are used in an SQL WHERE clause.
   b. Select all source message elements with values that are relevant to the rows that are retrieved, and drag them from the Source pane into the Edit pane. These values can be matched against the selected database columns.
   c. Click **Edit** → **Content Assist** to open Content Assist.
   d. From Content Assist, select the functions to apply to message elements in the database call.

The following example of a $db:select entry shows a database column that is matched against a constant or a field from an input message:

```
$db:select_1.BROKER50.JDOE.RESOLVEASSESSOR.ASSESSORTYPE = 'WBI' or $db:select_1.BROKER50.JDOE.
RESOLVEASSESSOR.ASSESSORTYPE = $source/tns:msg_tagIA81CONF/AssessorType
```

A $db:select entry retrieves all qualifying rows, therefore more than one row might be retrieved. By default, the selection is treated as repeating, which is indicated by the ‘for' row below $db:select in the Spreadsheet pane.

After you have configured the $db:select, populate the target message from the database by dragging the database column from the Source Pane to the message element in the Target pane. The mapping is indicated by a line between the database column in the Source pane and the element in the Target pane. An entry for this map in XPath format also appears in the Spreadsheet pane. Triangular icons are displayed in the Source and Target panes next to objects that have been mapped.

You can map to an element within a View, the name of which is annotated with `(read-only view)`.

Using database selects

By default, a $db:select entry is accompanied by a 'for' row that iterates over the select result set. Ensure that your 'for' row is in the correct position for your mapping. The behavior of the map is determined by the position of the 'for' row in the Spreadsheet pane. For example, if the results of the $db:select statement matched five rows in the database, and the 'for' row is the parent of the $target entry in the Spreadsheet pane, five complete messages are generated by the mapping node. If the 'for' row is positioned within the message body, one message is generated with five repeating elements in the message body.

A mapping can contain multiple 'for' rows, associated with a $db:select entry, that perform a single database select and iterate over the results multiple times. For example, multiple 'for' rows can be used in conditional mappings, where an individual 'for' row is used with a 'condition' or an 'else'.

A 'for' row is not always required and can be deleted in the following circumstances:

- If the database select returns only one row.
- If you use an aggregate XPath function on the select results.
  For example: fn:sum or fn:count.

All $db:select expressions must be within the scope of the $db:select entry in the Spreadsheet pane, meaning that each one must be a descendant of the select statement. If a $db:select expression is out of scope, the Message Mapping editor moves the $db:select entry to a position where the $db:select expression is in scope. Ensure that the position of the $db:select entry is correct for your message mapping.

Database table join

Database table join is supported for tables within the same database. For example, consider the following two tables where PRODUCT_ID and PART_NUMBER match.

```
Table    Column        Row 1    Row 2    Row 3      Row 4
ORDER    PRODUCT_ID    456      456      345        123
         QUANTITY      100      200      300        400

PRODUCT  PART_NUMBER   123      456      789        012
         PART_NAME     pen      pencil   paperclip  glue
         PRICE         0.25     0.15     0.02       0.99
```

A $db:select expression with the following syntax joins the tables:

```
$db:select.MY_DB.SCHEMA1.ORDER.PRODUCT_ID=$db:select.MY_DB.SCHEMA2.PRODUCRT.PART_NUMBER
```

The $db:select expression in the example generates the following result set.

```
                Row 1    Row 2    Row 3
PRODUCT_ID      456      456      123
QUANTITY        100      200      400
PART_NUMBER     456      456      123
PART_NAME       pencil   pencil   pen
PRICE           0.15     0.15     0.25
```

You can then use the 'for' row to iterate through the results set in the same way as results from a single table.

Mapping a target element from database stored procedures:

Use a DB2 or Oracle database stored procedure as a source in a Mapping node.

Before you start:

You must include the database stored procedure in your database definition. See “Adding database definitions to the workbench” on page 578. Only DB2 and Oracle databases are supported for this task.

To map a target element from a database stored procedure, set up the Mapping node to:

- run the stored procedure to retrieve the relevant rows from the database
- populate the message target elements with values from the database

Complete these steps to map a target element from a stored procedure:

1. Add a stored procedure as a source for the mapping, as described in “Adding a database as a source or target” on page 575. After you have added a stored procedure to the mapping, the Source pane and the Spreadsheet pane contain $db:proc entries. These $db:proc entries contain details of the parameters used by the stored procedure. For each parameter it shows its name, its mode (either IN, INOUT, or OUT), and its data type.

The $db:proc entry in the Spreadsheet pane also contains a 'for' row. By default, the position of the 'for' row in the $db:proc entry indicates that when there is more than one record in the result set, the output message body contains repeating elements. If you want a separate message to be output for each record in a result set, move the $target element in the Spreadsheet pane below the 'for' row.

If you are using a DB2 stored procedure which specifies a maximum number of result sets, the Source pane also includes a list of result sets. If you are using an Oracle stored procedure which specifies ref cursor parameters of mode INOUT or OUT, the Mapping node treats them as named result sets rather than parameters. The Mapping node does not support ref cursor parameters of mode IN.

2. If your stored procedure has parameters of mode IN or INOUT provide values for each of them:
   a. Type an expression in the **Value** field next to the parameter in the Spreadsheet pane.
   b. Select the parameter in the Spreadsheet pane, drag an element from the source message to the Edit pane, and press Enter.
3. If your stored procedure sets a return value, right-click on the $db:proc entry in the Source pane and select **Toggle Add/Remove Stored Procedure Return Value**. An element called ReturnValue of type INTEGER is added to the Source pane.

   DB2 on distributed systems returns SQLCODE if the stored procedure does not set a return value.

   DB2 on z/OS and Oracle stored procedures do not set a return value.
4. For each result set that your stored procedure returns, define the columns it contains:
   a. Right-click on the result set entry in the Source pane and select **Add or Remove Result Set Columns**. The Add or Remove Result Set Columns window opens.
   b. (Optional) Select an item from the **Restore previous settings from** list. The **Result set columns** list is replaced.
   c. (Optional) Select columns in the **Available database table columns** list, and add them to the **Result set columns** list.
   d. (Optional) Type a column name in the **New column** field, and add it to the **Result set columns** list. Use this method to add calculated columns that are named in the stored procedure.
   e. (Optional) Change the name in the **Save current settings to** field that is used to save the columns shown in the **Result set columns** when you click **Finish**. The settings are saved in the `\.metadata` directory. You can retrieve these settings in another message map, as described in step 4a. If you do not change the default name the settings are automatically retrieved when the stored procedure is used in another message map.

Mapping a target element from database user-defined functions:

Use an Oracle database user-defined function as a source in a Mapping node.

Before you start:

You must include the database user-defined function in your database definition. See “Adding database definitions to the workbench” on page 578. Only Oracle databases are supported for this task.

To map a target element from a database user-defined function, set up the Mapping node to:

- run the user-defined function to retrieve the relevant rows from the database
- populate the message target elements with values from the database

Complete these steps to map a target element from a user-defined function:

1. Add a user-defined function as a source for the mapping, as described in "Adding a database as a source or target" on page 575. After you have added a user-defined function to the mapping, the Source pane and the Spreadsheet pane contain $db:func entries. These $db:func entries contain details of the parameters used by the user-defined function. For each parameter it shows its name, its mode (either IN, INOUT, or OUT), and its data type.

   The $db:func entry in the Spreadsheet pane also contains a 'for' row. By default, the position of the 'for' row in the $db:func entry indicates that when there is more than one record in the result set, the output message body contains repeating elements. If you want a separate message to be written for each record in a result set, move the $target element in the Spreadsheet pane below the 'for' row.

   If you are using an Oracle user-defined function which specifies ref cursor parameters of mode INOUT or OUT, the Mapping node treats them as named result sets rather than parameters. The Mapping node does not support ref cursor parameters of mode IN.

   If you are using an Oracle user-defined function which returns a value of ref cursor data type, the Mapping node treats it as a named result set.
2. If your user-defined function has parameters of mode IN or INOUT provide values for each of them:
   a. Type an expression in the **Value** field next to the parameter in the Spreadsheet pane.
   b. Select the parameter in the Spreadsheet pane, drag an element from the source message to the Edit pane, and press Enter.
3. For each result set that your user-defined function returns, define the columns it contains:
   a. Right-click on the result set entry in the Source pane and select **Add or Remove Result Set Columns**. The Add or Remove Result Set Columns window opens.
   b. (Optional) Select an item from the **Restore previous settings from** list. The **Result set columns** list is replaced.
   c. (Optional) Select columns in the **Available database table columns** list, and add them to the **Result set columns** list.
   d. (Optional) Type a column name in the **New column** field, and add it to the **Result set columns** list. Use this method to add calculated columns that are named in the user-defined function.
   e. (Optional) Change the name in the **Save current settings to** field that is used to save the columns shown in the **Result set columns** when you click **Finish**. The settings are saved in the`\.metadata` directory. You can retrieve these settings in another message map, as described in step 3a. If you do not change the default name the settings are automatically retrieved when the user-defined function is used in another message map.

Deleting data from a database with a mapping node:

You can use a DataDelete or Mapping node to delete data from a database, based on information from an input message, another database or both.

You must do the following before you can delete data from a database using a mapping node:

1. Create a message flow project
2. Create a message flow
3. Define message flow content that includes a DataDelete or a Mapping node
4. Create a message map file from a DataDelete node or Create a message map file from a Mapping node

You cannot create mappings to delete data from a database by dragging from the source to the target. Instead, you select rows to delete based on the content of the source. You can use an expression to match the content of the source to the target field, for example, use the following instructions to delete all rows in the database that match the content of a field from the input message:

1. Right-click your DataDelete or Mapping node, and click **Open Map**. The "Message Mapping editor" on page 1530 opens with your selected sources and targets.
2. Select `$db:delete` in the Spreadsheet pane.
3. Drag the appropriate source element from the message in the Source pane to the Edit pane. For example, `$source/shipTo/accNum`.
4. Drag the appropriate target database field from the Target pane to the Edit pane. For example, `$db:delete.SAMPLE.MYSCHEMA.CUSTOMER.CONTACT_ID`.
5. Change the expression in the Edit pane to set the target field to be equal to the source element. For example, `$source/shipTo/accNum = $db:delete.SAMPLE.MYSCHEMA.CUSTOMER.CONTACT_ID`.

You can use conditional mappings such as If statements to create more complex mappings that define which data to delete from a database. You can also use conditional statements in a Mapping node to perform different database operations depending on the content of the input message. For example, you can add a Table Inserts target, a Table Updates target and a Table Deletes target to a message map, then use conditional statements to define which of the operations to perform.

Creating a database to database mapping:

You can create a message map that uses a database as both the source and target. The contents of the source database can be used to interact with the same or a different database table. The message map can also include a message as a source, but a message is not required. You can, for example, use a timer node to schedule regular updates to a database.

Before creating a message map file with a database to database mapping, ensure you do the following:

1. Create a message flow project
2. Create a message flow
3. Define message flow content
4. Create database definitions

To create a database to database mapping:

1. Right-click a node that supports database mapping in your flow, such as the Mapping node, and click **Open Map**. The New Message Map wizard opens for your node.
2. Select the Data Sources and any Messages that you want to use as sources for your map from **Select map sources**.

   If you cannot find the Messages or Data Sources that you expect, clear the **Apply working set filtering to artifact selection(s) on the page** check box.
3. From Select map targets expand the database operation that you want to perform. You can select from the following database operations:
   - Table Inserts
   - Table Updates
   - Table Deletes
4. Select the database tables that you want to map.

   You can create a message map that performs a combination of database inserts, updates or deletes by selecting database tables from different database operations. For example, if you want to create a conditional mapping that updates data in a database if it already exists, but inserts the data if it does not already exist in the database, you can select the same database table under Table Inserts and Table Updates.
5. Select **OK** to create the new message map. The "Message Mapping editor" on page 1530 opens with the selected sources and targets.

After you have created a message map file, you can now configure the message mappings.

Storing a BLOB message in a database table using a message map:

Use the Message Mapping editor to create a bit stream from a BLOB message, and store it in a database table.

Before you start:

Create a mapping; see "Creating a message map file from a Mapping node" on page 552.

Take the following steps:

1. In the Target pane, right-click the column that will store the bitstream, and select **Enter Expression** from the menu.
2. In the Edit pane, type `esql:asbitstream()`. You can use content assist; `asbitstream` is a *Field function*.
3. Drag the source field, for example `$source/po:purchaseOrder`, to the Edit pane, placing it between the parentheses. The entry in the Edit pane looks like this:

   ```
   esql:asbitstream($source/po:purchaseOrder, 'purchaseOrder',
   'PurchaseOrder', 'XML1', 0, 0, $esql:FolderBitStream)
   ```

   Alternatively, you can use content assist to select the `esql:asbitstream` function. In the Edit pane press Ctrl+Space to display a list of available functions and associated parameters. The `asbitstream` function is an ESQL Field function. The function can take other parameters; see "Predefined ESQL mapping functions" on page 1544.

   When you move the cursor out of the Edit pane, or press Enter, the mapping is displayed between the fields in the Source and Target panes.

Mapping from a BLOB field in a database table to an output message:

Use the Message Mapping editor to parse a bit stream from a field in a database table into a folder in a target message.

Before you start:

Create a mapping; see "Creating a message map file from a Mapping node" on page 552.

Take the following steps:

1. Right-click the element in the target pane, and select **Enter Expression** from the menu.
2. In the Edit pane, type `msgmap:element-from-bitstream()`.
3. Drag the field from the database table to the Edit pane, placing it between the parentheses, for example:

   ```
   msgmap:element-from-bitstream($db:select.RESERVDB.USER.XMLFLIGHTTB.FLIGHTDATE)
   ```

   Alternatively, you can use content assist to select the `msgmap:element-from-bitstream` function. In the Edit pane press Ctrl+Space to display a list of available functions and associated parameters. The function can take other parameters; see "Predefined mapping functions" on page 1551. For example:

   ```
   msgmap:element-from-bitstream($db:select.RESERVDB.USER.XMLFLIGHTTB.FLIGHTDATE,
   'ReserveMessageSet', 'FlightMessage', 'XML1', 0, 0, $esql:FolderBitStream)
   ```

   When you move the cursor out of the Edit pane, or press Enter, the mapping is displayed between the fields in the Source and Target panes.

Creating and calling submaps and subroutines

Use submaps, ESQL subroutines, or both, to map source elements to target elements.

The following topics describe how to work with submaps and ESQL subroutines:

Creating a new submap:

This topic describes how to create a new submap. There are three ways to create a new submap:

- **Using File → New → Message Map**
  1. From the Broker Application Development perspective, click **File → New → Message Map**. The New Message Map wizard opens.
  2. Specify the project name and the name for the new submap.
  3. Specify that the new map is a submap by selecting the option: **Submap called by another map**.

4. Select the combination of Message Components or Data Sources that you want to use as sources for your map from **Select map sources** and select the combination of Message Components or Data Targets that you want to use as targets for your map from **Select map targets**.

   If you cannot find the Message Components, Data Sources or Data Targets that you expect, clear the **Apply working set filtering to artifact selection(s) on the page** check box.
5. Click **Finish**.

The new submap opens in the Message Mapping editor.

- **Using Create new submap**
  1. From the Broker Application Development perspective, open the message map for the required node.
  2. In the Source pane, expand the tree and select the source.
  3. In the Target pane, expand the tree and select the target.
  4. Right-click either the source or target, then click **Create New Submap**.

  The new submap opens in the Message Mapping editor. If the original map file was called `simple_mapping.msgmap`, the new submap is called `simple_mapping_submap0.msgmap`.
- **Using Convert to submap**
  1. From the Broker Application Development perspective, open the message map.
  2. Select one of the following types of submap to create a new submap from an inline mapping:
     - Element statement that maps a global element or an element of global type
     - Attribute statement that maps either a global attribute or an attribute of a global type
     - Database insert statement
     - Database update statement
     - Database delete statement
  3. Right-click the mapping statement that you want to convert to a submap or database submap in the Script pane, and click **Convert to submap**. A new submap is created and a statement is added to the original message map to call the new submap.

  The new submap opens in the Message Mapping editor. If the original map file was called `simple_mapping.msgmap`, the new submap is called `simple_mapping_submap0.msgmap`.

Creating a new submap for a wildcard source:

You can map a wildcard value in the source to a wildcard value in the target.

You might expect a wildcard in a Mapping node for example, when you are using a SOAP message (where the Body element contains a wildcard). This type of wildcard represents the payload of the message, where the payload is a message that is defined elsewhere in the message set. The submap can involve from 0 to n source wildcards and 0 or 1 target wildcards.

The "Message Mapping editor" on page 1530 shows three kinds of wildcard, all of which allow you to create a submap:

Mapper construct	Message model construct	Choose concrete item for submap
Wildcard element	Wildcard element	Global element
Wildcard attribute	Wildcard attribute	Global attribute
Message with Wildcard Message child	Group with Composition of Message and Content Validation of Open or Open Defined	Message

1. Switch to the Broker Application Development perspective.
2. Open the message map for the required node.
3. In the Source pane, expand the tree and select the source wildcard.
4. In the Target pane, expand the tree and select the target wildcard.
5. Right-click either the source or the target wildcard, and click **Create new submap**. The Wildcard Specification wizard opens.
6. From the Wildcard Specification wizard, select the concrete item that will replace the source wildcard, according to the values shown in the table at the beginning of this topic.
7. Click **Next**.
8. From the Wildcard Specification wizard, select the concrete item that will replace the target wildcard, according to the values shown in the table at the beginning of this topic.
9. Click **Finish**.
10. Click **OK**. The submap opens in the Message Mapping editor.
11. From the submap, map the source message elements to the target message elements as required.
12. Click **OK**.

Creating a submap to modify a database:

Use the Create New Database Submap wizard to create a submap to modify a database.

You must have an existing message map from which to call the submap. The following steps describe how to create a submap to modify a database:

1. In the Broker Application Development perspective, open the calling message map.
2. In the Source pane, right-click the message component containing the fields to be used to modify the database and click **Create New Database Submap**. The source can be a wildcard, an element, or an attribute. The Create New Database Submap wizard opens.
3. If the selected source is a wildcard, select a message or message component for the source wildcard from **Select a defined item to replace the source wildcard** pane. If you cannot find the message components that you expect, clear the **Apply working set filtering to artifact selection(s) on the page** check box.
4. From **Select database submap targets** expand the database operation that you want to perform. You can select from the following database operations:
   - **Table Inserts**
   - **Table Updates**
   - **Table Deletes**

5. Select the database tables that you want to map. If you cannot find the Data Targets that you expect, select the **Show all resources in workspace** check box.
6. Click **OK**. A new submap is created with the selected message or message component in the Source pane, and the database table in the Target pane. In the calling message map `$db:call` is added to the Target pane.

After you have created the submap file, configure the message mappings for the database table.

Converting a message map to a submap:

You can convert between a message map and a submap in order to change the usage of the map. You might convert a message map to a submap because you want to reuse the same mappings for multiple nodes. Use the following instructions to convert a message map to a submap for each message in the message map.

1. From the Broker Application Development perspective right-click your message map and click **Open**.
2. Right-click `$source` in the Source pane and select **Add or Remove Headers and Folders**. The Add or Remove Headers and Folders dialog opens.
3. Select **No folders (map body element only)**. Any previously selected headers or folders are cleared.
4. Click **OK** to remove the headers and folders.
5. Repeat steps 2 to 4 to select to map body element only from your target message under `$target` in the Target pane.
6. Delete target map statements for existing mappings to properties, message headers or other folders such as LocalEnvironment. These mappings are flagged with warning messages after the headers are removed.
7. Remove the reference to the new submap from any mapping nodes. If a reference to the submap exists in the Mapping Routine property of a mapping node an error message is displayed on the message flow.
8. Save the submap, and check for any broken references as indicated by errors or warnings in the Problems view.

The submap is now ready to be used. See calling a submap for more information.

To convert a submap to a message map, click **Add or Remove Headers and Folders** for the source and target messages, and select to map **Selected headers**. You must ensure that no other maps call the changed map, check for errors in the Problems view to indicate this problem. See mapping headers and folders for more information about mapping headers, Properties and the LocalEnvironment.

Converting an inline mapping to a submap:

To change the usage of a map, you can convert between inline mappings in a message map and a submap.

You might convert parts of an existing message map to a submap because you want to reuse the same mappings for multiple nodes. You can convert inline mappings to submaps from messages or databases. You must select one of the following types of statement to create a submap or a database submap:

- Element statement that maps a global element or an element of global type

- Attribute statement that maps either a global attribute or an attribute of a global type
- Database insert statement
- Database update statement
- Database delete statement

The target that is added to the new submap is the global element, attribute, type, or database insert, update, or delete that you select. The source that is added to the new submap is the appropriate global element or type from the source from the selected mappings that are included in the submap. If mappings included in the selected statement do not reference any source, only a target is added to the submap. If the source contains a database select that is not referenced by any other part of the original message map, a database select is added as a source to the submap, and removed from the original message map. However, if the source contains a database select that is referenced by any other part of the original message map, the original message map retains the select, and the submap performs a separate select. If you do not want to perform two database select operations, do not use a database submap under these conditions.

Use the following instructions to convert inline mappings in a message map to a submap or database submap:

1. From the Broker Application Development perspective right-click your message map and click **Open**.
2. Right-click the mapping statement that you want to convert to a submap or database submap in the Script pane, and click **Convert to submap**. A new submap is created and a statement is added to the original message map to call the new submap.

The submap is now ready to be used.

Calling a submap:

Use the Call Existing Submap wizard to call a submap. The submap must already be in the workspace.

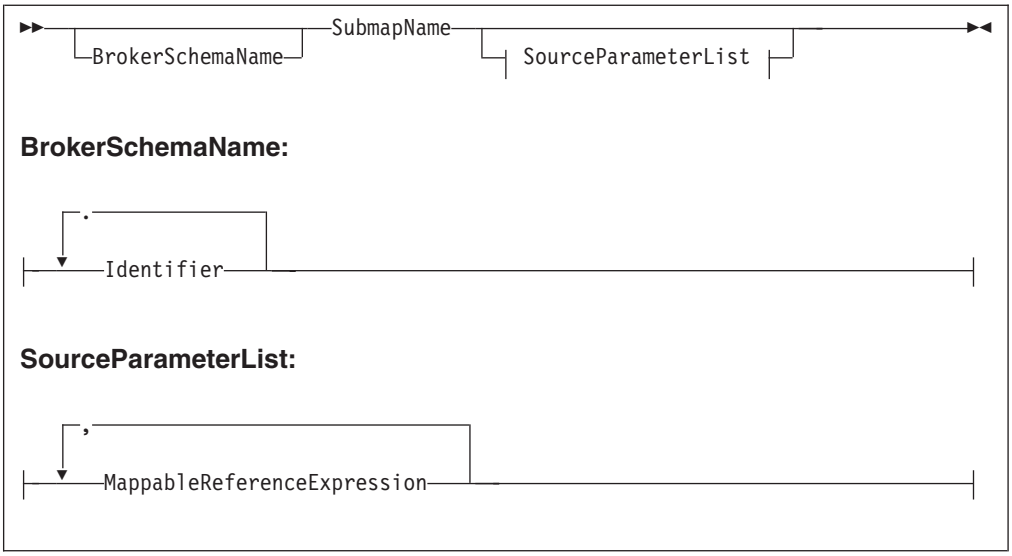
If a submap does not exist, use the **Create New Submap** menu option to create a submap that you can call. This action creates the new submap in the same folder as the calling map. It also allocates a default map operation name to the new submap. If the source or target in the calling map is a wildcard, a wizard allows you to choose a replacement element.

You can also map from a wildcard to a wildcard.

The following steps describe how to call a submap:

1. In the Broker Application Development perspective, open the calling map.
2. In the Source and Target panes, select one or more sources and one target. Any of the sources or the target can be a wildcard, an element, or an attribute.
3. Click **Map** → **Call Existing Submap**. The Call Existing Submap wizard opens.
4. Complete the wizard, following the on-screen instructions.

The call to the submap takes the following format:



Only source parameters appear in the call and only message parameters appear in the list.

Calling a map from ESQL:

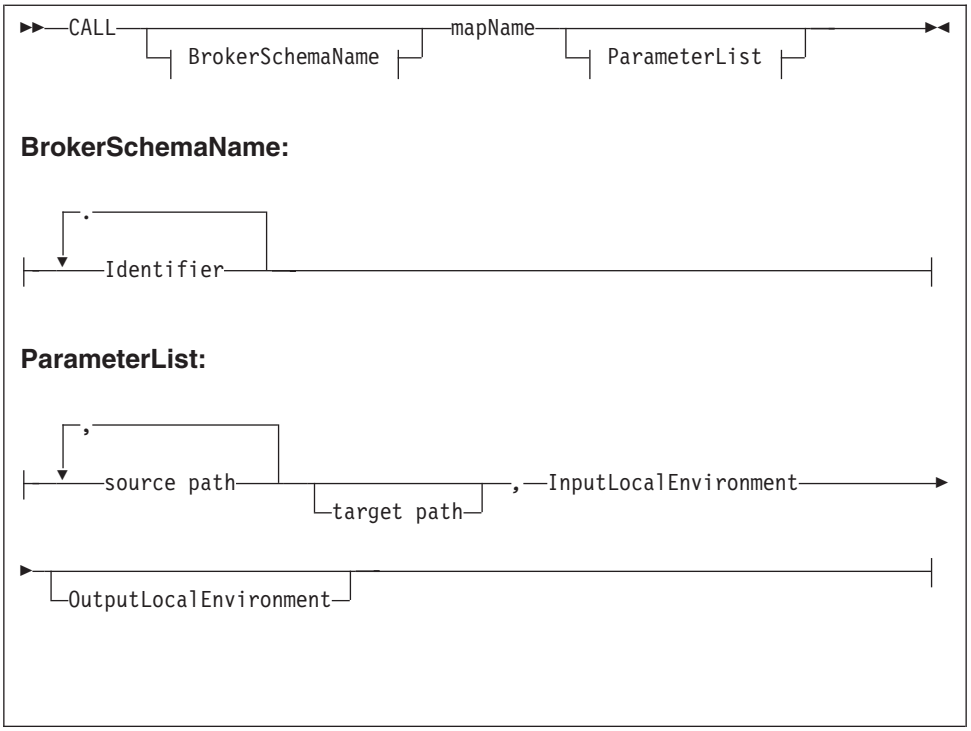
You can use the Message Mapping editor to perform mappings to a certain level of complexity. To create even more complex mappings, use ESQL. ESQL is particularly suitable for interacting with databases.

If a map does not already exist, create one.

The following steps describe how to call a map (which can be a submap) from ESQL. Calling a map from ESQL uses different parameters to when you call a submap from another map due to this extra level of complexity (when calling a map from ESQL, the two local environment parameters are added at the end of the CALL statement).

1. Switch to the Broker Application Development perspective.
2. Right-click a node that supports ESQL and click **Open ESQL**. The ESQL file opens for that node.
3. Add a CALL statement to call a map. Alternatively, press Ctrl+Space to access ESQL content assist, which provides a drop-down list that includes the map name and expected parameters.

   The following syntax diagram demonstrates the CALL statement:



Notes:

1. Only source parameters appear in the call and only message parameters appear in the list.
2. If the map builds a message target, include the `target path` and `OutputLocalEnvironment` parameters. If the map does not build a message target (for example, if the map interacts with a database), these two parameters do not appear.

Calling an ESQL routine:

To call an existing ESQL routine from a mapping, select the routine from the Call Existing ESQL Routine wizard. The ESQL routine must already exist in the workspace.

1. Switch to the Broker Application Development perspective.
2. Open the required mapping.
3. In the Source pane, select the required source.
4. In the Target pane, select the required target.
5. Right-click either the Source or Target pane and click **Call ESQL Routine**. The Call ESQL routine wizard opens.
6. Select the routine where the parameters and return types match the source and target selection.
7. Click **OK**.

Creating and calling your own user-defined ESQL routine:

For complex mappings, you can create user-defined ESQL functions that can be called from the Message Mapping editor.

This topic describes how to create a user-defined ESQL function, and how to use it in an existing message map.

1. Switch to the Broker Application Development perspective.
2. Create a new ESQL file, or open an existing ESQL file.
3. Enter your ESQL function in the ESQL file. Ensure that you do not enter the ESQL in any existing modules.
4. Save the ESQL file.
5. Right-click your Mapping node node, and click **Open Map** to open your message map in the Message Mapping editor.
6. Select the target that you want to generate using your ESQL function from the appropriate target message or target database table.
7. In the Edit pane, enter the expression to call the ESQL function and any parameters to pass to the function. For example:

   ```
   esql:concatValues($source/Pager/Text, ' Powered by IBM.')
   ```

   Where `concatValues` is the name of the user-defined ESQL function and the following parameters:

   - `$source/Pager/Text` is a field in the source message
   - `' Powered by IBM.'` is text

   The following code is the ESQL used for the user-defined ESQL function in the preceding example:

   ```
   CREATE FUNCTION concatValues(IN val INTEGER, IN str CHAR) RETURNS CHAR
    BEGIN
     return str || ' plus int val ' || CAST(val AS CHAR);
    END;
   ```

   You can also use **Edit** → **Content Assist** to select user-defined ESQL functions. The user-defined ESQL functions are located at the end of the list of ESQL functions.
8. Save the message map file by clicking **File** → **Save**.

Calling a Java method:

To call an existing Java method from a mapping node, select the method from the Call Existing Java Method wizard, or enter an XPath expression in the Edit pane.

See "Message mapping tips and restrictions" on page 600 for information about the types of methods that are available through the wizard, and through content assist.

Using the wizard:

To use the Call Existing Java Method wizard, take the following steps:

1. Switch to the Broker Application Development perspective.
2. Open the required message map.
3. If the method requires input parameters, select one or more fields in the Source pane. Your choice determines which methods are subsequently displayed in the wizard. If you select no source fields, the wizard shows only methods that take no parameter. If you select two fields, the wizard displays only methods that take two parameters, and so on.
4. In the Target pane, select the required target field to be mapped to the return value of the Java method. The target field must be a simple, non-wildcard type.
5. Right-click either the Source or Target pane and click **Call Java Method**. The Call Existing Java Method wizard opens.

6. Select the method, then click **OK**.

Entering an XPath expression:

You can enter the expression directly, without using content assist. Enter a function call expression, with the following syntax:

```
java:package_name.class_name.method_name (parameters)
```

You can omit the package name if there is no package, or if you are using a default package.

To use content assist, take the following steps:

1. Switch to the Broker Application Development perspective.
2. Open the required message map.
3. In the Edit pane, click **Edit → Content Assist**.
4. Select **java: (Java Methods)**, then click **Edit → Content Assist**. All qualifying Java methods are displayed.
5. Select the method.
6. If the method requires input parameters, drag the appropriate source fields to the method's parameter area. The number of source fields included must match the number of input parameters that the method takes.

This is an example of a method that takes one input parameter:

```
java:mypackage1.MyClass1.myMethod1($source/po:purchaseOrder/po:comment)
```

Separate parameters by a comma:

```
java:mypackage1.MyClass1.myMethod1($source/po:purchaseOrder/name,
  $source/po:purchaseOrder/phone)
```

Transforming a SOAP request message

SOAP is an XML-based language defined by the W3C for sending data between applications. A SOAP message comprises an envelope containing:

- An optional header (containing one or more header blocks)
- A mandatory body.

For common envelope message formats, such as SOAP, where both the envelope and the messages that can appear within that envelope have to be modeled, use the Message Mapping editor to select from available messages at points in the model that are defined with **Composition="message"** and **Content validation="open"** or **"open defined"**.

Define the mappings by selecting from the allowed constituent messages. For example, in the case of SOAP, the outer level message is called Envelope and has one mandatory child element called Body, which is modeled with **Composition="message"**. If the permitted content of Body is modeled by separate messages Msg1 ... MsgN, define mappings for each separate message (Envelope.Body.Msg1 to Envelope.Body.MsgN).

For complex type elements with type composition message, the Message Mapping editor follows these rules:

Content validation	Messages offered
Closed	Messages available in any message sets in the workspace
Open defined	Any message defined within the current message set
Open	The Message Mapping editor does not support open or open defined content when the type composition is NOT message

Mapping an embedded message

When you are working with type composition message, with content open or open-defined (and no children defined), map the embedded message using a submap:

1. In the main map, expand the levels (both source and target) of Envelope and Body until you find the wildcard message, and select this on both the source and target sides.
2. Right-click either the source or target and click **Create New Submap**.
3. From the dialog box, select a source (for example `reqmess`) and a target (for example `rspmess`).
4. With the submap open in the Message Mapping editor, make the appropriate mappings between the source (`reqmess`) and target (`rspmess`).

Editing a default-generated map manually

A map that is generated by the Message Mapping editor might not do everything that you want; you can change maps manually to enhance its operation.

You can edit the structure directly by inserting, moving, copying, pasting, and deleting rows. The pop-up menu provides a list of available editing actions with their keyboard equivalents. Here are some specific operations that you might want to perform:

- "Creating message headers"
- "Creating conditional mappings"

Creating message headers:

When you create a map from a Mapping node, or if you select the option **Message map called by a message flow node** from the New Message Map wizard, the map that is created allows additional elements including WebSphere MQ, HTTP, and JMS headers to be mapped.

If you use a Mapping node for a database to message mapping without specifying a source message, the Message Mapping editor cannot generate an output header for the map file that is created. To ensure that an output header is created, perform one of the following steps:

- When you create the message map, add message headers to the target message and ensure that all mandatory fields in the header are set.
- Add an additional source message to the map. The source message must be the same message as the intended target message. You do not need to create any mappings from the source message because the headers from the source message are automatically copied to the output message tree.

Creating conditional mappings:

When a mapping involves one of the following items:

- Schema choice group
- Derived type element
- Substitution group member
- Wildcard character
- Repeating element

the default mapping that is generated by the Message Mapping editor might be placed under an `if` statement. If the `if` statement is not what you had expected, edit the statements; here are the changes that you can make:

- Move statements in or out of an `if/elseif/else` block.
- Reorder `if` and `elseif` statements.
- Create new `elseif` statements.
- Create new `if` statements.

See the "Configuring conditional mappings" on page 570 topic for more information about conditional mappings.

:

Message mapping tips and restrictions

Information to help you use message mapping.

These tips assume that you have created a Mapping node within the message flow, opened the Message Mapping editor, and selected both a source and a target message:

- "Mapping a message when the source is a list and the target is a list from source, but with a new entry at the top of the list"
- "Changing the target message domain" on page 601
- "Overriding the database schema name" on page 601
- "Mapping batch messages" on page 602
- "Mapping restrictions" on page 602
- "Requirements for calling a Java method" on page 602

Mapping a message when the source is a list and the target is a list from source, but with a new entry at the top of the list

1. Expand the target to display the element for which you want to create a new first instance. This might be a structure or a simple element.
2. Right-click the element and click **If**. An `if` line appears and wraps around the element.
3. Right-click the `if` line and click **Else If**. There are now two entries in the spreadsheet for your element.
4. Set the first of these entries to values of your choice.
5. Right-click the second entry and click **For**. A `for` line appears in the spreadsheet.
6. Set the second entry to the value or values mapped from the source.
7. Set the `for` entry to the looping condition.
8. Click **For**, then drag the source field that represents the loop condition to the Expression editor.

Changing the target message domain

When you first create a mapping, you nominate a message set for the target message. The parser that is associated with the output message is determined by the Message Domain property of the message set. For example, when a message set is first created, the default message domain is MRM. Therefore, the Mapping node generates ESQL with the following format:

```
SET OutputRoot.MRM.Fielda...
```

If you change the runtime parser to XMLNSC, for example, the Mapping node generates ESQL with the following format:

```
SET OutputRoot.XMLNSC.MessageA.FieldA...
```

The parser of the source message is determined by the contents of the MQRFH2 header or by the properties of the input node. The Mapping node generates a target message with a parser that matches the message domain of the message set. The Mapping node supports the following message domains:

MRM
XMLNSC
XMLNS
MIME
SOAP
DataObject
JMSMap
JMSStream
XML
BLOB
IDOC

To change the message domain property of your message set:

1. Open the message set file `messageset.mset`.
2. Change the Message Domain property to a supported domain.
3. Save your message set, and save any message flows and message maps that reference your message set, if they have not already been saved. Saving these files generates updated ESQL for mapping the changed message set.

   If you have made no updates to your flows or message maps after changing the message domain of your message set, you must clean the related message flow projects so that updated ESQL code can be generated:

   a. Select a project and click **Project** → **Clean Project**.
   b. Select **Clean all projects** or **Clean selected projects**.
   c. Click **OK**.
4. Deploy the changed message set.
5. Deploy the message flow that contains the mappings, and test your ESQL in a Compute node and in other nodes to ensure that the message flow still functions as expected.

Overriding the database schema name

To change the database schema name that is generated in ESQL, use the Override Database Schema wizard in the Specify Runtime Schema dialog box. The default name is the schema name of the database definitions that are imported into the

workbench. Use the Specify Runtime Schema dialog box to change the value.

Mapping batch messages

You can configure a message mapping that sorts, orders, and splits the components of a multipart message into a series of batch messages. These components can be messages or objects, and they can have different formats; in this case, each component is converted and the message is reassembled before being forwarded.

1. Use a "RouteToLabel node" on page 1145 in the message flow to receive multipart messages as input.

   The RouteToLabel node is the next node in sequence after the "Mapping node" on page 1052, and causes the flow to jump automatically to the specified label. You can specify a single RouteToLabel value in a splitting map for all maps that output a message assembly. You can also use conditions to set the RouteToLabel value, depending on the values in the source message.
2. Use the "Message Mapping editor" on page 1530 to build maps that transform and propagate batch messages using a single node, without having to define an intermediate data structure.

Multipart messages can also contain repeating embedded messages, where each repeated instance of a message is propagated separately. Embedded messages must be from the same message set as the parent message.

Mapping restrictions

Unless stated explicitly, you can achieve the required functionality by calling an ESQL function or procedure. The following restrictions apply:

- Mixed content fields cannot be mapped.
- Exceptions cannot be thrown directly in Mapping nodes.
- Self-defined elements cannot be manipulated in Mapping nodes (support for wildcard symbols is limited if the wildcard symbols represent embedded messages).
- The Environment tree cannot be manipulated in the Mapping node.
- User variables cannot be defined or set.
- CASE expressions cannot be emulated; you must use IF ... ELSE.
- Trees cannot be copied from input to output in order to modify elements within the copied tree. For example, the following ESQL cannot be modeled in a Mapping node:

  ```
  SET OutputRoot.MQMD = InputRoot.MQMD; SET OutputRoot.MQMD.ReplyToQ = 'NEW.QUEUE';
  ```

  You must set each field in the structure individually if you intend to modify one or more sibling fields.

Requirements for calling a Java method

All of the following conditions must be satisfied for the method to be shown in the Call Existing Java Method wizard, or in content assist:

- The method must be public static, in a Java project.
- The default scope of the search is all methods in .java source files in the workspace, excluding application libraries and application JAR files in the Java build path, and all the JRE system libraries. To change the scope, change the preferences in the Toolkit:

  1. Click **Window** → **Preferences**.

2. Expand the **Broker Development** node, then click **Message Map Editor**.
3. Select and clear the check boxes as appropriate.

- The method must have a return value.
- Return values and Java parameters must be one of the following data types:

Data types	Equivalent XML schema type for mapping	Comments
java.lang.Long	byte, unsignedShort, long, unsignedByte, short, int, unsignedInt	
java.lang.Double	float, double	
java.math.BigDecimal	nonNegativeInteger, negativeInteger, integer, nonPositiveInteger, positiveInteger, unsignedLong, decimal	
java.lang.String	NCName, Name, IDREF, normalizedString, string, anyURI, NOTATION, token, NMTOKEN, language, ID, ENTITIES, QName, ENTITY	
byte[]	base64Binary, hexBinary	
com.ibm.broker.plugin.MbDate	date, gYear, gMonth, gDay, gYearMonth, gMonthDay	
com.ibm.broker.plugin.MbTime	time	
com.ibm.broker.plugin.MbTimestamp	dateTime	
java.lang.Boolean	boolean	
com.ibm.broker.plugin.MbElement	A complex type	Valid only for input parameters, not return values.

- The method must not have a `throws` clause.
- The number of source fields selected must match the number of input parameters that the method takes.
- If you are using a working set, only the methods in the current working set are displayed. Clear the check box **Apply working set filtering to artifact selection(s) on the page** to display all the methods in the workspace. If you are not using a working set, all the methods in the entire workspace are displayed. Content assist shows all methods in the workspace, whether you are using a working set or not.

When you create the BAR file you must select the Java project or JAR file that contains the method that you are calling.

Message mapping scenarios

This section contains some message mapping scenarios that demonstrate how to make the most of the message mapping functions:

- "Scenario A: Mapping an airline message" on page 604
- "Scenario B: Simple message enrichment" on page 612
- "Scenario C: Using a broker as auditor" on page 618
- "Scenario D: Complex message enrichment" on page 623
- "Scenario E: Resolving a choice with alternative message data" on page 641
- "Scenario F: Updating the value of a message element" on page 642

Scenario A: Mapping an airline message

This scenario demonstrates how to create, configure, and deploy a new message mapping.

The message flow that is used in this example reads an XML input message (an airline message), then uses a Mapping node to achieve the following transformations:

- Convert the input message from XML to COBOL
- Modify an element in the input message from the results that are obtained by a database lookup
- Concatenate two elements in the input message to form a single element in the output message

Follow these steps to complete this scenario:

1. "Example names and values"
2. "Connect to the database and obtain the definition files" on page 606
3. "Create the message flow" on page 607
4. "Create the mapping file" on page 607
5. "Configure the mapping file" on page 608
6. "Deployment of the mapping" on page 611

Next: Go to "Example names and values."

Example names and values:

View the resources that are created for the airline message scenario.

The following table describes the names and definitions of the resources that are created.

Resource	Name	Definition
Alias name	AIRLINEDBALIAS	The same as connection name and database name in this case
Broker archive (BAR) file name	AIRLINE	Contains the message flow and message set projects, and the mapping file, and is deployed to the default execution group for the run time
COBOL copybook	AirlineRequest.cbl	Controls the structure of the COBOL output message
Connection name	AIRLINECONN	The same as alias name and connection name in this case
Database	AIRLINEDB	Contains the table XREF and is the same as the connection name and the alias name in this case
Database table (table tree)	XREF	Contains lookup information (in this case the two-code airline city code abbreviations STATE=Illinois, ABBREV=IL)
Default project	AIRLINE_MFP	The default message flow project. You copy the database definitions to this project.

Resource	Name	Definition
Default queue manager	WBRK6_DEFAULT_QUEUE_MANAGER	The default queue manager that controls the message queue
ESQL select operation	`$db:select.AIRLINEDB.AIRLINE_SCHEMTREE. XREF.ABBERV`	The ESQL select operation that performs a qualified database select operation
Input (XML) message	`c:\airline\data\AirlineRequest.xml`	The input message (in this case an XML message)
Input message source fields	FirstName,LastName	The source elements in the input message that are concatenated
Input queue name property	AIRLINE_Mapping_IN	The input queue
Mapping node rename	XML_TO_COBOL	The name of the node in the message flow that performs the mapping (the node was renamed from its default name)
Message mapping file name	AIRLINE.msgmap	The file that contains the mapping configuration used by the Mapping node
Message Set property	AIRLINE_MSP2	The message set project name
Message Type property	msg_AIRLINEREQUEST	The message type
Message Format	Binary1	The custom wire format (CWF) for COBOL output message
Message flow name	AIRLINE_Mapping	The name of the message flow
Message flow project	AIRLINE_MFP	The name of the message flow project
Message set projects	AIRLINE_MSP1,AIRLINE_MSP2	The names of the message set projects
Msg Domain node property	MRM	The message domain node property
Msg Set Name node property	AIRLINE_MSP1	The message set name node property
Msg Type property	AirlineRequest	The message type property
Msg Format node property	XML1	The input message format
Output message target field	NAME	The result of concatenating FirstName and LastName in the input message. NAME is the element that is created in the output message.
Output queue name property	AIRLINE_Mapping_OUT	The output queue name
Resource folder	airline\resources	The folder where the mapping resources are stored
Schema tree	AIRLINE_SCHEMTREE	The name of the schema tree
Source	ABBREV	The source
Source tree	$source/AirlineRequest	The source tree

Resource	Name	Definition
Source message	AirlineRequest	The source message
Target	STATE	The target
Target message	AIRLINEREQUEST	The target message
Target tree	$target/AIRLINEREQUEST	The target tree
XPath concatenation function	`fn:concat(fn:concat($source/ AirlineRequest/Purchase/ Customer/FirstName,' '), $source/AirlineRequest/ Purchase/Customer/LastName)`	The W3C XPath 1.0 Specification function that concatenates FirstName and LastName

Now go to "Connect to the database and obtain the definition files."

Connect to the database and obtain the definition files:

Demonstrate how to define a database connection to enable a message flow to access a database table. You must define the database to the workbench.

Before you start:

Create a message flow project.

The Database Definition wizard uses the JDBC interface to communicate with the database server; therefore, you must ensure that the database client JAR file on your local or workbench host is at a compatible level to allow communication with the database server. If necessary, consult your database vendor for further advice.

1. Switch to the Broker Application Development perspective.
2. Select the message flow project that you want to create the database definition files in, and click **File** → **New** → **Database Definition**. The New Database Definition File wizard opens.
3. Select an existing data design project, or click **New** to create a new data design project.
4. Select the database type and version that you want to connect to from the **Database and Version** list. Ensure that you select a database from the list that is supported by WebSphere Message Broker; you can use this wizard in a shell-share environment with other Rational products that support other databases or versions.

   For a list of databases supported by the broker, see Supported databases.
5. Click **Next**.
6. Select **To create a new database connection**, and click **Next**.
7. Clear **Use default naming convention**, and enter a connection name; for example, AIRLINEDBALIAS.
8. Enter values for the Connection to the database; for example, Database name, Host name, and Port number.
9. Enter values for the User ID and Password to connect to the database. Click **Test Connection** to verify the settings that you have selected for your database. The default Port number for a DB2 database is 50000. If the connection fails, enter other values such as 50001, 50002 and so on, for the Port number, and test the connection again.
10. Click **Next**.
11. Select one or more database schemas from the list, and click **Next**.

12. Click **Finish**.
13. Add the data design project as a reference to the message flow project:
    a. Right-click the message flow project, and click **Properties**.
    b. Click **Project References**, and select the data design project from the list to add as a referenced project.
    c. Click **OK**.

The database definition is added to a new Data design project. You have now defined the database to the mapping tools.

Now go to "Create the message flow."

Create the message flow:

Before you start:

1. Create a message flow project.
2. "Connect to the database and obtain the definition files" on page 606.
3. Create a message flow by adding an MQInput node, and renaming the node (for example, to AIRLINE_Mapping_IN).
4. Set the queue name property (for example, to AIRLINE_Mapping_IN).
5. Add an MQOutput node to the message flow, and rename the node (for example, to AIRLINE_Mapping_OUT).

This topic demonstrates how to specify a message flow project, add a Mapping node, wire the nodes, and set the node properties.

1. Switch to the Broker Application Development perspective.
2. Open the message flow (for example, AIRLINE_Mapping) within the message flow project (for example, AIRLINE_MFP). This message flow forms the starting point for the mapping task.
3. Open the palette of nodes and add a Mapping node to the message flow. You might need to scroll down to find the Mapping node.
4. Rename the Mapping node (for example, to XML_TO_COBOL) by right-clicking the node and clicking **Rename**.
5. Wire the node terminals (for example, AIRLINE_Mapping_IN> XML_TO_COBOL> AIRLINE_Mapping_OUT).
6. Modify the properties of the MQInput node (for example, AIRLINE_Mapping_IN) by right-clicking the node and clicking **Properties**.
7. Click **OK**.
8. Modify the properties of the Mapping node (for example, XML_TO_COBOL).
9. Set the data source as the database name (for example, AIRLINEDBALIAS)
10. Click **OK**.

You have now created the required message flow, wired the nodes, and set the node properties.

Now go to "Create the mapping file."

Create the mapping file:

Before you start:

Follow the instructions in these topics:

1. "Connect to the database and obtain the definition files" on page 606
2. "Create the message flow" on page 607

This topic demonstrates how to create a new mapping file, specify how it will be used, and specify the source and target mappable elements.

1. Switch to the Broker Application Development perspective.
2. From the message flow, right-click the mapping node (for example XML_TO_COBOL) and click **Open Map**. The New Message Map wizard opens.
3. Select the combination of Messages, Data Sources or both that you want to use as sources for your map from **Select map sources** (for example, AirlineRequest) from the first message set project (for example, AIRLINE_MSP1). If you cannot find the Messages, Data Sources or Data Targets that you expect, clear the **Apply working set filtering to artifact selection(s) on the page** check box.
4. Select the combination of Messages, Data Targets or both that you want to use as targets for your map from **Select map targets** (for example, AIRLINEREQUEST) from the second message set project (for example, AIRLINE_MSP2).
5. Click **Finish**.

You have now created the mapping file, defined its usage, and specified the source and target mappable elements.

Now go to "Configure the mapping file."

Configure the mapping file:

Before you start:

Follow the instructions in these topics:

1. "Connect to the database and obtain the definition files" on page 606
2. "Create the message flow" on page 607
3. "Create the mapping file" on page 607

This set of topics demonstrates how to configure the mapping file by:

1. Specifying a data source
2. Mapping the message properties
3. Writing an XPath function that concatenates the elements in the input message
4. Specifying an ESQL select command

Now go to "Specify the data source."

Specify the data source:

Before you start:

Follow the instructions in these topics:

1. "Connect to the database and obtain the definition files" on page 606
2. "Create the message flow" on page 607
3. "Create the mapping file" on page 607

This topic demonstrates how to specify the database to use as the source for the mapping.

1. From the Message Mapping editor Spreadsheet pane, select an item. The item that you select determines the scope of the $db:select entry that is created by the action. For example, you can select $target, an element, an attribute, a For condition, or another $db:select entry. The Select database as mapping source dialog box opens.
2. Right-click and click **Select data source**.
3. From the Select Database as mapping source page, select a database (for example, AIRLINEDB) and click **Finish**. The Message Mapping editor adds the sources of the database table (for example, the XREF table) to the tree in the Message Mapping editor Source pane.

You have now added the data source to the Message Mapping editor Source pane.

Now go to "Map the message properties."

Map the message properties:

This topic demonstrates how to map the message set, message type and message format properties.

Before you map the message properties, ensure you do the following:

1. "Connect to the database and obtain the definition files" on page 606
2. "Create the message flow" on page 607
3. "Create the mapping file" on page 607
4. "Specify the data source" on page 608

To map the message properties:

1. From the Message Mapping editor Spreadsheet pane, expand the entries by clicking **+** to reveal the message properties.
2. Right-click $target and click **Insert Children**.
3. Right-click Properties and click **Insert Children**. The MessageSet, MessageType and MessageFormat properties contain default values for the target message.
4. To change the format of the output message, change the MessageFormat property to the appropriate value. You must use quotation marks around the value of MessageFormat, because the values are string literals and without quotation marks, the values will be interpreted as XPath locations.
5. From the Message Mapping editor Source pane, expand the properties for the $source tree, and for each remaining property, map the source element to its corresponding target element by dragging from source to target. Alternatively, select Properties in both the source and the target panes and use **Map by Name** to map all of the properties.
6. Save the map by clicking **File → Save**.

You have now mapped message set, message type and message format properties.

Now go to "Add the XPath concatenate function."

Add the XPath concatenate function:

Before you start:

Follow the instructions in these topics:

1. "Connect to the database and obtain the definition files" on page 606
2. "Create the message flow" on page 607
3. "Create the mapping file" on page 607
4. "Specify the data source" on page 608
5. "Map the message properties" on page 609

This topic demonstrates how to write an XPath function that concatenates the FirstName and LastName from the input message, and adds a white space separator in the target NAME element. When you add the XPath expression and save the map, link lines are automatically generated between the source and target to indicate that these elements are mapped.

1. From the Message Mapping editor Source pane, select the first source to concatenate (for example, FirstName), Ctrl+click to select the second source to concatenate (for example, LastName), and drag both elements onto the target (for example, NAME) in the Target pane.
2. From the Message Mapping editor Spreadsheet pane, select the target (for example, NAME).
3. From the Edit pane, enter the XPath function (for example, `fn:concat($source/AirlineRequest/Purchase/Customer/FirstName, ' ', $source/AirlineRequest/Purchase/Customer/LastName)`
4. Save the map by clicking **File** → **Save**.

You have now added an XPath function that concatenates the two source elements in the input message into a single target element in the output message.

Now go to "Add the database Select operation."

Add the database Select operation:

Before you start:

Follow the instructions in these topics:

1. "Connect to the database and obtain the definition files" on page 606
2. "Create the message flow" on page 607
3. "Create the mapping file" on page 607
4. "Specify the data source" on page 608
5. "Map the message properties" on page 609
6. "Add the XPath concatenate function" on page 609

This topic provides instructions on how to add a database select operation that makes a qualified selection from a data source. In the spreadsheet pane the `$db:select` statement has the default value `fn:true()`, which returns all entries in the table. You must therefore replace this value with one that qualifies the selection, for example:

```
$db:select.LAB13STA.ARGOSTR.XREF.STATE=$source/AirlineRequest/Purchase/Customer/State
```

The XPath in this example selects only the records from the database where the value in the STATE column for each record matches the value of the State field from the input message. In the spreadsheet pane the `$db:select` statement is associated with a `For` entry which is used to iterate over the mappings for the

target message. For each row in the database matching the `$db:select` statement a separate target message is created with the mappings beneath `$target`.

The following steps describe how to create message mappings to generate a target message based on records in a database matching the contents of an input message:

1. In the spreadsheet pane replace the existing value `fn:true()` with the value to match in the database (for example a field in the input message as shown in the preceding example).
2. Create mappings from the database fields in the Source pane to include in the target message, by dragging them from the source pane onto the target elements. A `$db:select` statement is added to the value column in the spreadsheet pane (for example, `$db:select.AIRLINEDB.AIRLINE_SCHEMTREE.XREF.ABBREV`).
3. Create any mappings you require from the source message to the target message.
4. Save the mapping by clicking **File** → **Save**.
5. Save the message flow.
6. Check for any errors in the Problems view.

You have now made a qualified selection from the database.

Now go to "Deployment of the mapping."

Deployment of the mapping:

How to map to the run time, creating a broker archive (BAR) file for deployment in the default execution group.

Before you start:

Follow the instructions in these topics:

1. "Connect to the database and obtain the definition files" on page 606
2. "Create the message flow" on page 607
3. "Create the mapping file" on page 607
4. "Configure the mapping file" on page 608

Create a broker archive (BAR) file, which contains the message flow, message set projects and the mapping file, for deployment to the run time and the default execution group.

1. From the Broker Administration perspective, right-click the project under the Broker Archives heading.
2. Click **New** → **Message Broker Archive**.
3. Name the BAR file (for example, AIRLINE).
4. Click **Add to**. The Add to broker archive page is displayed.
5. Select the message flow project and message set projects that are used by this flow (for example, AIRLINE_MFP,AIRLINE_MSP1, AIRLINE_MSP2) and click **OK**. The projects are added to the BAR file. A status indicator and message panel show when the process has completed.
6. Check to ensure that the required projects have been included in the BAR file.
7. Save the BAR file by clicking **File** → **Save**.

8. For deployment of the BAR file, right-click the BAR file and click **Deploy File**. The Deploy a BAR file page is displayed.
9. Select the default execution group, and click **OK**. A message in the Broker Administration message dialog box indicates successful deployment, and the deployed message flow project and message set projects are displayed in the Domains view. A message in the Event log also indicates successful deployment.

You have completed this scenario.

Scenario B: Simple message enrichment

This scenario demonstrates simple message enrichment. Use the Message Broker Toolkit to create message flows and message sets, and to create and deploy broker archive (BAR) files.

The scenario also involves creating a Configuration Manager and a broker, and inputting instance messages that can contain MQRFH2 headers.

This scenario uses repeating instances and requires the following mapping functions:

- MRM in, MRM out (non-namespace)
- Map a simple and complex element source - target
- Map the same element source - target
- Map a different element source - target
- Map an attribute source - target
- Map a one-sided element (edit mapping)
- Map a one-sided attribute (edit mapping)
- Perform arithmetic on numeric field mapping
- Map a repeating simple element - single instance
- Map all instances of a repeating simple element
- No MQRFH2 header

The names and values used for message flows, message sets, elements and attributes, and the expressions and code samples are for illustrative purposes only.

Follow these steps to complete this scenario:

1. "Develop a message flow and message model for simple and complex element mapping"
2. "Develop a message flow and message model for a target-only element" on page 614
3. "Develop a message flow and message model for dealing with repeating elements" on page 615
4. "Develop a message flow and message model for a simple message without an MQRFH2 header" on page 617

Develop a message flow and message model for simple and complex element mapping:

This is the first stage of the scenario to perform simple message enrichment. This topic demonstrates how to develop a message flow and message model for simple and complex element mapping, where there is the same source and target, a different source and target, or an attribute source and target. This task also involves changing field values and creating an instance document.

1. From the Broker Application Development perspective, create the following resources:
   a. a message set project (for more details, see Creating a message set).
   b. a message set called MAPPING3_SIMPLE_messages. Ensure that the message set is namespace enabled with XML wire format.
   c. a message definition file (no target namespace) called SIMPLE.
2. Create a message called addev1 that has the following structure:

```
addev1
        ssat          (xsd:string) local attribute
        ssel          (xsd:string) local element
        dsel1        (xsd:string) local element
        atel           local complex element
             latt       (xsd:string) attribute
        cel1          local complex element
             intel     (xsd:int) local element
             strel     (xsd:string) local element
        dsel2         (xsd:string) global element
        cel2           (cel2ct) global complex type
            intel        (xsd:int) local element
            fltel         (xsd:float) local element
```

3. Create a message flow project called MAPPING3_SIMPLE_flows.
4. Create a message flow called addev1 that contains the following mapping: MQInput -> Mapping -> MQOutput.
5. Open the map in the Message Mapping editor and select message addev1 as both source and target
6. Expand all levels of both messages and wire the elements as shown:

```
ssat --- ssat
ssel --- ssel
dsel1 -- dsel2
latt ---- latt
cel1 --- cel1
dsel2 -- dsel1
(cel2)
   intel ---- fltel
   fltel ---- intel
```

7. In the Spreadsheet pane, set the following expression:

```
dsel1  |  esql:upper($source/addev1/dsel2)
@latt  |   esql:upper($source/addev1/atel/@latt)
(cel2)
    intel   |  $source/addev1/cel2/fltel + 10
    fltel    |  $source/addev1/cel2/intel div 10
```

8. Create an instance document with the appropriate RFH2 header and the following data:

```
<addev1 ssatt="hello">
<ssel>this</ssel>
<dsel1>first</dsel1>
<atel latt="attrib"/>
<cel1>
<intel>2</intel>
<strel>lcomp</strel>
</cel1>
<dsel2>second</dsel2>
<cel2>
<intel>252</intel>
<fltel>3.89E+1</fltel>
</cel2>
</addev1>
```

You have created the following resources:

- message set MAPPING3_SIMPLE_messages, which you have populated with message addev1
- message flow addev1 in project MAPPING3_SIMPLE_flows, which contains the mapping addev1_Mapping.msgmap
- a file that contains an instance message

Now deploy the message set and message flow.

Deploy the message set and message flow:

This is the second stage of the scenario to perform simple message enrichment. This topic demonstrates how to deploy the message set and message flow and run the data through the broker.

1. Create a broker archive (BAR) file called addev1.
2. Add the message set MAPPING3_SIMPLE_messages and the message flow addev1 to the BAR file.
3. Deploy the BAR file to the broker.
4. Put the instance document on the input queue.

The output message looks like this:

```
<addev1 ssat="hello">
<ssel>this</ssel>
<dsel1>SECOND</dsel1>
<atel latt="ATTRIB"/>
<cel1>
<intel>2</intel>
<strel>lcomp</strel>
</cel1>
<dsel2>first</dsel2>
<cel2>
<intel>48</intel>
<fltel>2.5E+1</fltel>
</cel2>
</addev1>
```

Now go to "Develop a message flow and message model for a target-only element."

Develop a message flow and message model for a target-only element:

Before you start

Perform the steps in the following topic:

1. "Develop a message flow and message model for simple and complex element mapping" on page 612

This is the third stage of the scenario to perform simple message enrichment. This topic demonstrates how to develop a message flow and message model for a target-only element. It also involves attributing a mapping and creating an instance document.

1. Create a message called addev2, which has the following structure:

```
addev2
    matt            (xsd:string) local attribute
    ssel             (xsd:string)  local element
    csel             local complex element
        elatt         (xsd:string)  local attribute
```

2. Create a second message called trigger, which has the following structure:

```
trigger
   start              (xsd:string) local element
```

3. Create a message flow called addev2, which contains the following mapping: MQInput -> Mapping -> MQOutput.
4. Open the map and select trigger as the source and addev2 as the target.
5. In the Spreadsheet pane, expand the target message fully and set the target fields as shown:

```
matt    |   'first attribute'
ssel     |   'string element'
elatt    |   'second attribute'
```

6. Expand the Properties folder in the Spreadsheet pane and set the following value:

```
MessageType   |   'addev2'
```

7. Create an instance document with the appropriate RFH2 header and the following data:

```
<trigger>
<start>yes</start>
</trigger>
```

You have created the following resources:

- two messages called addev2 and trigger
- a message flow called addev2, which contains the mapping addev2_Mapping.msgmap
- a file that contains an instance message

Now deploy the message set and message flow.

Deploy the message set and message flow:

This is the fourth stage of the scenario to perform simple message enrichment. This topic demonstrates how to deploy the message set and message flow and run the data through the broker.

1. Create a broker archive (BAR) file called addev2.
2. Add the message set MAPPING3_SIMPLE_messages and the message flow addev2 to the BAR file.
3. Deploy the BAR file to the broker.
4. Put the instance document on the input queue.

The output message looks like this:

```
<addev2 matt="first attribute">
<ssel>string element</ssel>
<csel elatt="second attribute"></csel>
</addev2>
```

Now go to “Develop a message flow and message model for dealing with repeating elements.”

Develop a message flow and message model for dealing with repeating elements:

Before you start

Perform the steps in the following topics:

1. "Develop a message flow and message model for simple and complex element mapping" on page 612
2. "Develop a message flow and message model for a target-only element" on page 614

This is the fifth stage of the scenario to perform simple message enrichment. This topic demonstrates how to develop a message flow and message model for dealing with repeating elements, a single instance and all instances.

1. Create a message called addev3, which has the following structure:

```
addev3
    frepstr        (xsd:string) local element, minOcc=3, maxOcc=3
    vrepstr       (xsd:string) local element, minOcc=1, maxOcc=4
    urepstr       (xsd:string)  local element, minOcc=1, maxOcc=-1
```

2. Create a message flow called addev3, which contains the following mapping: MQInput -> Mapping -> MQOutput.
3. Open the map and select addev3 as both source and target
4. In the upper pane, map each source to the corresponding target, as illustrated in this example:

```
frepstr --- frepstr
vrepstr --- vrepstr
urepstr --- urepstr
```

5. In the Spreadsheet pane, expand fully the target addev3.
6. Highlight and delete the For item above the vrepstr entry.
7. Create an instance message with the appropriate RFH2 header and the following data:

```
<addev3>
<frepstr>this</frepstr>
<frepstr>that</frepstr>
<frepstr>other</frepstr>
<vrepstr>only one</vrepstr>
<vrepstr>extra</vrepstr>
<urepstr>first</urepstr>
<urepstr>second</urepstr>
<urepstr>third</urepstr>
<urepstr>fourth</urepstr>
<urepstr>fifth</urepstr>
</addev3>
```

You have created the following resources:

- a message called addev3
- a message flow called addev3, which contains the mapping addev3_Mapping.msgmap
- a file that contains an instance message

Now deploy the message set and message flow.

Deploy the message set and message flow:

This is the sixth stage of the scenario to perform simple message enrichment. This topic demonstrates how to deploy the message set and message flow and run the data through the broker.

1. Create a broker archive (BAR) file called addev3.
2. Add the message set MAPPING3_SIMPLE_messages and the message flow addev3 to the BAR file.
3. Deploy the BAR file to the broker.

4. Put the instance document on the input queue.

The output message looks like this:

```
<addev3>
<frepstr>this</frepstr>
<frepstr>that</frepstr>
<frepstr>other</frepstr>
<vrepstr>only one</vrepstr>
<urepstr>first</urepstr>
<urepstr>second</urepstr>
<urepstr>third</urepstr>
<urepstr>fourth</urepstr>
<urepstr>fifth</urepstr>
</addev3>
```

Now go to "Develop a message flow and message model for a simple message without an MQRFH2 header."

Develop a message flow and message model for a simple message without an MQRFH2 header:

Before you start

You must complete the following tasks:

1. "Develop a message flow and message model for simple and complex element mapping" on page 612
2. "Develop a message flow and message model for a target-only element" on page 614
3. "Develop a message flow and message model for dealing with repeating elements" on page 615

This is the seventh stage of the scenario to perform simple message enrichment. This topic demonstrates how to develop a message flow and message model for a simple message without an MQRFH2 header.

1. Create a message set called MAPPING3_SIMPLE_xml. A message set project is also automatically created; this message set project has the same name as the message set that you created.
2. On the message set parameters page, set the *Message Domain* property to XML.
3. Create a message definition file called SIMPLE.
4. Create a message called addev4 that has the following structure:

```
addev4
   str1                  (xsd:string)  local element
   cel                    local complex element
       int1               (xsd:int) local element
       bool1             (xsd:boolean) local element
```

5. Create a message flow called addev4 that contains the following connected nodes: MQInput -> Mapping -> MQOutput.
6. Select the Input Message Parsing properties tab of the MQInput node, and set the *Message Domain* property to XML.
7. Open the map and select addev4 as both source and target.
8. Map the inputs to the corresponding outputs, as shown in this example:

```
str1 --- str1
int1 --- int1
bool1 --- bool1
```

9. Create an instance message that has no MQRFH2 header, but contains the following data:

```
<addev4>
<str1>this</str1>
<cel>
<int1>452</int1>
<bool1>0</bool1>
</cel>
</addev4>
```

You have created the following resources:

- A message set called MAPPING3_SIMPLE_xml that contains the message addev4
- A message flow called addev4 that contains the mapping addev4_Mapping.msgmap
- A file that contains an instance message

Now deploy the message set and message flow.

Deploy the message set and message flow:

This is the final stage of the scenario to perform simple message enrichment. This section describes how to deploy the message set and message flow, and how to the run the data through the broker.

1. Create a broker archive (BAR) file called addev4.
2. Add the message flow called addev4 to the BAR file.
3. Deploy the BAR file to the broker.
4. Put the instance document on the input queue.

The output message has the following content:

```
<addev4>
<str1>this</str1>
<cel>
<int1>452</int1>
<bool1>0</bool1>
</cel>
</addev4>
```

You have completed the scenario.

Scenario C: Using a broker as auditor

This scenario demonstrates how to use a broker as an auditor. Use the Message Broker Toolkit to create message flows and message sets, and to create and deploy broker archive (BAR) files.

It also involves creating a Configuration Manager and a broker, and sending instance messages that can contain MQRFH2 headers.

The scenario uses database updates that have been defined by using mappings. The broker receives a confirmation for a provisional booking, the message flow inserts a row into a database table representing the confirmation, updates a counter in another table representing the key of the confirmation, and deletes the provisional booking from a third table.

This scenario uses the DataDelete, DataInsert and DataUpdate nodes in the message flow, and requires the following mapping functions:

- Mapping in DataInsert node
- Combine input data into single insert
- Mapping in DataUpdate node
- Mapping in DataDelete node
- BAR file to override data source

The names and values used for message flows, message sets, elements and attributes, and the expressions and code samples are for illustrative purposes only.

Follow these steps to complete this scenario:

1. "Develop a message flow"
2. "Deploy the message set and message flow" on page 621
3. "Override the data source of one of the nodes" on page 621
4. "Create a BAR file, edit the configuration, and deploy" on page 622

Develop a message flow:

Create a message flow to use in mapping scenario C, which shows how to use your broker as an auditor.

This step is the first task in creating "Scenario C: Using a broker as auditor" on page 618. Develop a message flow to map several fields of input data into a single insert record for a database. See also how to update another table, delete a third table, and develop corresponding message models and instance messages.

1. Create a database called MAPDB, and a table called CONFIRMATION, which contains the following columns:

   ```
   RESID                INTEGER
   ```

2. Populate the CONFIRMATION table with the value shown:

   ```
   9052
   ```

3. Create another table called RESERVATION, which contains the following columns:

   ```
   RESID           INTEGER
   NAME            VARCHAR(20)
   PARTY           INTEGER
   PAYMENT         DECIMAL(8,2)
   ```

4. Populate the RESERVATION table with the values shown:

   ```
   8214,'ARCHIBALD',2,0.0
   2618,'HENRY', 4, 120.0
   9052,'THAW', 3, 85.0
   ```

5. Create another table called PROVISIONAL, which contains the following columns:

   ```
   RESID                  INTEGER
   ```

6. Populate the PROVISIONAL table with the values shown:

   ```
    8214 2618
   ```

7. Create a Windows ODBC Data Source Name for the database, then create a definition for the database in the workbench by clicking **File → New → Database Definition File**. For information about how to complete this step, including how to choose a supported database and version, follow the instructions provided in "Adding database definitions to the workbench" on page 578.

8. Create a message set project and a message set called MAPPING3_AUDIT_messages (ensuring that the message set is namespace enabled, with XML wire format) and create a message definition file called AUDIT.
9. Create a message called addev1, which has the structure:

```
addev1
   id                (xsd:int) local element
   status            (xsd:string) local element
   name              (xsd:string) local element
   size              (xsd:int) local element
   payment           (xsd:decimal) local element
```

10. Create a message flow project called MAPPING3_AUDIT_flows.
11. Create a message flow called addev1, which contains the following structure: MQInput ->DataInsert -> DataUpdate -> DataDelete -> MQOutput.
12. For the DataDelete node, set the Data Source property to MAPDB.
13. Open the mapping for the DataInsert node and select MAPPING3_AUDIT_messages addev1 as the source, and MAPDB.SCHEMA.CONFIRMATION as the target.
14. Wire the source to the target as shown:

```
addev1             MAPDB
       id -------------- RESID
```

15. For the DataUpdate node, set the Data Source property to MAPDB.
16. Open the mapping for the DataUpdate node and select MAPPING3_AUDIT_messages addev1 as the source, and MAPDB.SCHEMA.RESERVATION as the target.
17. Wire the source to the target as shown:

```
addev1             MAPDB
       id -------------- RESID
       name ---------- NAME
       size ------------ PARTY
       payment ------- PAYMENT
```

18. In the Message Mapping editor Spreadsheet pane, select $db:update and change fn:true() to $db:update.MAPDB.MQSI.RESERVATION.RESID = $source/addev1/id and $source/addev1/status = 'CONFIRM'.
19. For the DataDelete node, set the Data Source property to MAPDB.
20. Open the mapping for the DataDelete node and select MAPPING3_AUDIT_messages addev1 as the source, and MAPDB.SCHEMA.PROVISIONAL as the target.
21. In the Message Mapping editor Spreadsheet pane, select $db:delete and change fn:false() to $db:delete.MAPDB.MQSI.PROVISIONAL.RESID = $source/addev1/id.
22. Create the following instance message with appropriate MQRFH2 headers:

```
<addev1>
<id>8214</id>
<status>CONFIRM</status>
<name>ARCHIBALD</name>
<size>2</size>
<payment>1038.0</payment>
</addev1>
```

You have created the following resources:

- A message set called MAPPING3_AUDIT_messages, which is populated with the message addev1

- A message flow called addev1 in project MAPPING3_AUDIT_flows, which contains the mapping files `addev1_DataInsert.msgmap`, `addev1_DataUpdate.msgmap`, and `addev1_DataDelete.msgmap`
- The database MAPDB with populated tables CONFIRMATION, RESERVATION, and PROVISIONAL
- A file that contains an instance message for test.

Next: Continue with the next step, "Deploy the message set and message flow."

Deploy the message set and message flow:

Deploy the message flow that you have created to support mapping scenario C, which shows how to use your broker as an auditor.

Before you start

Perform the first step for this scenario, described in "Develop a message flow" on page 619.

This step is the second task in creating "Scenario C: Using a broker as auditor" on page 618. Deploy the message set and message flow, and run the instance messages through the broker.

1. Create a BAR file called `addev1`.
2. Add the message set MAPPING3_AUDIT_messages, and the message flow addev1, to the BAR file.
3. Deploy the BAR file to the broker.
4. Put the instance document on the input queue.

The output messages are the same as the input messages. The database table has the following content:

```
CONFIRMATION
RESID
-----------
       9052
       8214

RESERVATION
RESID       NAME                 PARTY       PAYMENT
----------- -------------------- ----------- ----------
       8214 ARCHIBALD                      2    1038.00
       2618 HENRY                          4     120.00
       9052 THAW                           3      85.00

PROVISIONAL
RESID
-----------
       2618
```

Next: Continue with the next step, "Override the data source of one of the nodes."

Override the data source of one of the nodes:

Override the data source property for a node in the message flow that you have created to support mapping scenario C, which shows how to use your broker as an auditor.

Before you start

Complete the first two steps for this scenario:

1. "Develop a message flow" on page 619
2. "Deploy the message set and message flow" on page 621

This step is the third task in creating "Scenario C: Using a broker as auditor" on page 618. Override the data source of one of the nodes by changing the configuration of its broker archive (BAR) file.

1. Create a database called ALTDB, and a table called CONFIRMATION, which contains the following columns:

   ```
   RESID                INTEGER
   ```

2. Create a Windows ODBC Data Source Name for the database, then register the database in the workbench by clicking **File → New → Database Definition File**. For information about how to complete this step, including how to choose a supported database and version, follow the instructions provided in "Adding database definitions to the workbench" on page 578.

You have created a database called ALTDB with a table called CONFIRMATION.

Next: Continue with the next step, "Create a BAR file, edit the configuration, and deploy."

Create a BAR file, edit the configuration, and deploy:

Create a broker archive file and deploy the message flow for mapping scenario C, which shows how to use your broker as an auditor.

Before you start

Perform the first three steps for this scenario:

1. "Develop a message flow" on page 619
2. "Deploy the message set and message flow" on page 621
3. "Override the data source of one of the nodes" on page 621

This step is the fourth and last task in creating "Scenario C: Using a broker as auditor" on page 618. Learn how to create a broker archive (BAR) file, edit the configuration, and deploy.

1. Add the message flow addev1 to the BAR file again.
2. Select the Manage and Configure tab of the BAR file editor.
3. Expand the message flow, and click the DataInsert icon.
4. In the Properties view, change the Data Source field from MAPDB to ALTDB, and save the BAR file.
5. Deploy the BAR file to the broker.
6. Put the instance document on the input queue.

The output message is the same as the input. In the ALTDB database, the table has the following content:

```
CONFIRMATION
RESID
-----------
       8214
```

Next: You have now completed scenario C. If you want more information about how to use maps, return to "Message mapping scenarios" on page 603, and explore another scenario.

Scenario D: Complex message enrichment

This scenario demonstrates complex message enrichment and uses complex message manipulation. Use the Message Broker Toolkit to create message flows and message sets, and to create and deploy broker archive (BAR) files.

The scenario also involves creating a Configuration Manager and a broker, and putting instance messages that can contain MQRFH2 headers to input queues.

This scenario requires the following mapping functions:

- MRM in, MRM out (namespace)
- Other nodes required to complete message
- Conditional mapping
- CASE mapping (both syntax formats)
- If/condition
- Combining multiple source fields into a single target field (inter namespace)
- Nested repeating complex and simple elements
- Target data derived from database
- String, numeric, datetime functions
- User-defined ESQL procedures and functions
- User-defined Java routines

The names and values used for message flows, message sets, elements, and attributes, and the expressions and code samples, are for illustrative purposes only.

Follow these steps to complete this scenario:

1. "Develop a message flow that contains other nodes"
2. "Develop a message flow to map target fields from multiple other fields" on page 626
3. "Develop a message flow and message model for mapping a complex nested, repeating message" on page 628
4. "Develop a message flow for populating a target from a database" on page 634
5. "Develop a message flow using a user-defined ESQL function" on page 636
6. "Develop a message flow using a user-defined Java procedure" on page 638

Develop a message flow that contains other nodes:

Create a message flow to use in mapping scenario D, which describes complex message enrichment.

This step is the first task in creating "Scenario D: Complex message enrichment." Learn how to complete the following procedures:

- Develop a message flow that contains other nodes (for example, a Filter node)
- Create mappings that use conditions
- Develop corresponding message models, which use all main data types, and instance messages

1. Switch to the Broker Application Development perspective
2. Create the following resources:

- A message set project and a message set called MAPPING3_COMPLEX_messages, ensuring that the message set is namespace enabled with XML wire format
- A message definition file called COMPLEX, which has a target namespace www.complex.net, with prefix comp

3. Create messages addev1, addev1s and addev1n with the following structures:

```
addev1
   bool      (xsd:boolean) local element
   bin        (xsd:hexBinary) local element
   dat        (xsd:dateTime) local element
   dec       (xsd:decimal) local element
   dur        (xsd:duration) local element
   flt          (xsd:float) local element
   int         (xsd:int) local element
   str         (xsd:string) local element

addev1s
   bin        (xsd:hexBinary) local element
   dat        (xsd:dateTime) local element
   dur        (xsd:duration) local element
   str         (xsd:string) local element

addev1n
   dec       (xsd:decimal) local element
   flt          (xsd:float) local element
   int         (xsd:int) local element
```

4. Create a message flow project called MAPPING3_COMPLEX_flows.
5. Create a message flow called addev1 which contains the following structure:

```
MQInput -> Filter -> Mapping -> Compute
                  \                       \ --> RCD -> MQOutput
                   \-> Mapping1----------/
```

6. In the Filter node, set the following ESQL:

```
IF Body.bool THEN
            RETURN TRUE;
        ELSE
            RETURN FALSE;
        END IF;
```

7. In the Mapping node that is connected to the Filter node true terminal (Mapping1), open the map and select addev1 as source and addev1s as target.
8. Wire the source to target as shown:

```
bin --- bin
dat --- dat
dur --- dur
str --- str
```

9. In the Message Mapping editor Spreadsheet pane, expand Properties and set the following values:

```
MessageType   |     'addev1s'
```

10. Right-click the target dat and click **If**.
11. Replace the condition fn:true() with `$source/comp:addev1/str = 'dat'`.
12. Set the value for dat to `$source/comp:addev1/dat + xs:duration("P3M")`.
13. Right-click the condition and click **Else**.
14. Right-click the target dur and click **If**.
15. Replace the condition fn:true() with `$source/comp:addev1/str = 'dur'`.
16. Set the value for dur to `$source/comp:addev1/dur + xs:duration("P1Y")`.
17. Right-click the condition and click **Else**.
18. Open the map for the node that is connected to the false terminal (Mapping) of the Filter node and select addev1 as source and addev1n as target.

19. Wire the source to target as shown:

```
dec --- dec
flt --- flt
int --- int
```

20. In the Message Mapping editor Spreadsheet pane, expand Properties and set the following values:

```
MessageType     |     'addev1n'
```

21. Set the ESQL in the Compute node to:

```
CALL CopyMessageHeaders();
        SET OutputRoot.MRM.dec = InputBody.dec * 10;
        SET OutputRoot.MRM.flt = InputBody.flt * 10;
        SET OutputRoot.MRM.int = InputBody.int * 10;
```

22. In the ResetContentDescriptor node, set the Message Domain to XMLNS and select **Reset Message Domain**.

23. Create three instance messages with the appropriate MQRFH2 headers:

```
<comp:addev1 xmlns:comp="http://www.complex.net">
<bool>1</bool>
<bin><![CDATA[010203]]></bin>
<dat>2005-05-06T00:00:00+00:00</dat>
<dec>19.34</dec>
<dur>P2Y4M</dur>
<flt>3.245E+2</flt>
<int>2104</int>
<str>dat</str>
</comp:addev1>
```

```
<comp:addev1 xmlns:comp="http://www.complex.net">
<bool>1</bool>
<bin><![CDATA[010203]]></bin>
<dat>2005-05-06T00:00:00+00:00</dat>
<dec>19.34</dec>
<dur>P2Y4M</dur>
<flt>3.245E+2</flt>
<int>2104</int>
<str>dur</str>
</comp:addev1>
```

```
<comp:addev1 xmlns:comp="http://www.complex.net">
<bool>0</bool>
<bin><![CDATA[010203]]></bin>
<dat>2005-05-06T00:00:00+00:00</dat>
<dec>19.34</dec>
<dur>P2Y4M</dur>
<flt>3.245E+2</flt>
<int>2104</int>
<str>dat</str>
</comp:addev1>
```

You have created the following resources:

- A message set called MAPPING3_COMPLEX_messages, which is populated with the messages addev1, addev1s and addev1n
- A message flow called addev1 in the project MAPPING3_COMPLEX_flows, which contains the mapping files `addev1_Mapping.msgmap` and `addev1._Mapping1.msgmap`
- Files that contain instance messages for test

Now deploy the message set and message flow.

Deploy the message set and message flow:

This step is the second task in creating "Scenario D: Complex message enrichment" on page 623. Deploy the message set and message flow and run the instance messages through the broker.

1. Create a BAR file called addev1.
2. Add the message set MAPPING3_COMPLEX_messages and the message flow addev1 to the BAR file.
3. Deploy the BAR file to the broker.
4. Put the instance documents on the input queue.

The output messages have the following content:

```
<comp:addev1s xmlns:comp="http://www.complex.net">
<bin><![CDATA[010203]]></bin>
<dat>2005-08-06T00:00:00-01:00</dat>
<dur>P2Y4M</dur>
<str>dat</str>
</comp:addev1s>
```

Next: Continue with the next step, "Develop a message flow to map target fields from multiple other fields."

Develop a message flow to map target fields from multiple other fields:

Develop a message flow to map target fields from multiple other fields, and develop corresponding message models and instance documents to use in mapping scenario D, which describes complex message enrichment.

Before you start

Perform the first two tasks for this scenario, described in step "Develop a message flow that contains other nodes" on page 623.

This step is the third task in creating "Scenario D: Complex message enrichment" on page 623.

1. In the COMPLEX message definition, in namespace www.complex.net, create a message called addev2, which has the following structure:

```
addev2
   firstname            (xsd:string) local element
   lastname             (xsd:string) local element
   branch                (xsd:string) local element
   accountno           (xsd:string) local element
   balance               (xsd:decimal) local element
   transvalue            local complex element, base type xsd:decimal
      transdir            (xsd:string) local attribute
```

2. In the message set MAPPING3_COMPLEX_messages, create a new message definition file called COMP2, which has the target namespace www.comp2.net, with prefix c2.
3. In the COMP2 message definition, create a message called addev2out, which has the structure:

```
addev2out
    accountdetails      (xsd:string) local element
    transvalue            (xsd:decimal) local element
    balance                (xsd:decimal) local element
```

4. Create a message flow called addev2, which contains the following structure: MQInput -> Mapping -> MQOutput.
5. Open the map for the Mapping node, and select addev2 as the source and addev2out as the target.

6. Wire the source to target as shown:

   ```
   accountno --- accountdetails
   balance --- balance
   transvalue --- transvalue
   ```

7. In the Message Mapping editor Spreadsheet pane, expand Properties and set the following values:

   ```
   MessageType   |     'addev2out'
   ```

8. Set the accountdetails target to `fn:concat($source/comp:addev2/accountno, $source/comp:addev2/branch, $source/comp:addev2/lastname, $source/comp:addev2/firstname)`.
9. Right-click the target transvalue and click **If**.
10. Change the value of the `if` statement from fn:true() to `$source/comp:addev2/transvalue/@transdir = 'DEBIT'`.
11. Select transvalue and set its value to `$source/comp:addev2/transvalue * (-1)`.
12. Right-click the `if` statement and click **Else**.
13. Right-click the target balance and click **If**.
14. Change the value of the `if` statement from fn:true() to `$source/comp:addev2/transvalue/@transdir = 'DEBIT'`.
15. Select balance and set its value to `$source/comp:addev2/balance - $source/comp:addev2/transvalue`.
16. Right-click the `if` statement and click **Else If**.
17. Change the value of the `elseif` statement from fn:true() to `$source/comp:addev2/transvalue/@transdir = 'CREDIT'`.
18. Select balance following the second condition statement and set its Value to `$source/comp:addev2/balance + $source/comp:addev2/transvalue`.
19. Create two instance messages with the appropriate MQRFH2 headers:

    ```
    <comp:addev2 xmlns:comp="http://www.complex.net">
    <firstname>Brian</firstname>
    <lastname>Benn</lastname>
    <branch>52-84-02</branch>
    <accountno>567432876543</accountno>
    <balance>1543.56</balance>
    <transvalue transdir="DEBIT">25.28</transvalue>
    </comp:addev2>
    ```

    ```
    <comp:addev2 xmlns:comp="http://www.complex.net">
    <firstname>Brian</firstname>
    <lastname>Benn</lastname>
    <branch>52-84-02</branch>
    <accountno>567432876543</accountno>
    <balance>1543.56</balance>
    <transvalue transdir="CREDIT">25.28</transvalue>
    </comp:addev2>
    ```

You have created the following resources:

- A message called addev2 in the message definition called COMPLEX
- A message called addev2out in the message definition called COMP2
- A message flow called addev2, which contains the mapping file `addev2_Mapping.msgmap`
- Files that contain instance messages for test

Now deploy the message set and message flow

Deploy the message set and message flow:

This step is the fourth task in creating "Scenario D: Complex message enrichment" on page 623. Deploy the message set and message flow and run the instance messages through the broker.

1. Create a BAR file called addev2.
2. Add the message set MAPPING3_COMPLEX_messages and the message flow addev2 to the BAR file.
3. Deploy the BAR file to the broker.
4. Put the instance documents on the input queue.

The output messages have the following content:

```
<c2:addev2out xmlns:c2="http://www.comp2.net"  xmlns:comp="http://www.complex.net">
<accountdetails>567432876543 52-84-02 Benn Brian</accountdetails>
<transvalue>-25.28</transvalue>
<balance>1518.28</balance>
</c2:addev2out>
```

Next: Continue with the next step, "Develop a message flow and message model for mapping a complex nested, repeating message."

Develop a message flow and message model for mapping a complex nested, repeating message:

Develop a message flow and message model for mapping a complex nested, repeating message, and develop corresponding instance documents, to use in mapping scenario D, which describes complex message enrichment.

Before you start

Perform the previous tasks for this scenario, described in the following steps:

1. "Develop a message flow that contains other nodes" on page 623
2. "Develop a message flow to map target fields from multiple other fields" on page 626

This step is the fifth task in creating "Scenario D: Complex message enrichment" on page 623.

1. In the COMPLEX message definition, in namespace www.complex.net, create a message called addev3, which has the following structure:

```
addev3
 choice
    sstr                    (xsd:string) local element
    intrep                 (xsd:int) local element, minOcc=2, maxOcc=6
    dur                     (xsd:duration) local element
 choice
    comp1               local complex element
       dat1                (xsd:date) local element
       sval                (xsd:string) local element
    comp2               local complex element
       bool1              (xsd:boolean) local element
       dat2                (xsd:date) local element
    comprep            local complex element, minOcc=1, maxOcc=4
       int1                (xsd:int) local element
       dec1               (xsd:decimal) local element
 binel                     (xsd:hexBinary) local element
 lelem                    local complex element, base type xsd:string
    latt                    (xsd:int) local attribute
 lcomp                    local complex element
    head                    (xsd:string) local element
    incomp                local complex element
```

```
        count               (xsd:int) local element
        comp:gcompel   global complex element, minOcc=0, maxOcc=-1
           fstr                (xsd:string) local element
           multel             local complex element
              in1              (xsd:boolean) local element
              in2              (xsd:string) local element
              in3              (xsd:float) local element
     footer                  (xsd:string) local element
  repstr                      (xsd:string) local element, minOcc=1, maxOcc=-1
```

2. Create a message flow called addev3, which contains the following structure: MQInput -> Mapping -> MQOutput.
3. Open the map for the Mapping node, and select addev3 as the source and target.
4. Map each source element to its corresponding target element:

```
sstr --- sstr
intrep --- intrep
dur --- dur
dat1 --- dat1
sval --- sval
bool1 --- bool1
dat2 --- dat2
int1 --- int1
dec1 --- dec1
binel --- binel
lelem --- lelem
latt --- latt
head --- head
count --- count
fstr --- fstr
multel --- multel
footer --- footer
repstr --- repstr
```

5. In the Message Mapping editor Spreadsheet pane, for the `if` statement, change fn:true() to `fn:exists($source/comp:addev3/sstr)`.
6. For the `elseif` statement, change fn:true() to `fn:exists($source/comp:addev3/intrep)`.
7. For the second `elseif` statement, change fn:true() to `fn:exists($source/comp:addev3/dur)`.
8. For the first complex choice `if` statement, change fn:true() to `fn:exists($source/comp:addev3/comp1)`.
9. For the second complex choice `elseif` statement, change fn:true() to `fn:exists($source/comp:addev3/comp2)`.
10. For the third complex choice `elseif` statement, change fn:true() to `fn:exists($source/comp:addev3/comprep)`.
11. Create the following instance messages, with appropriate MQRFH2 headers:

```
<comp:addev3 xmlns:comp="http://www.complex.net">
<sstr>first</sstr>
<comp1>
<dat1>2005-06-24</dat1>
<sval>date value</sval>
</comp1>
<binel><![CDATA[3132333435]]></binel>
<lelem latt="24">twenty four</lelem>
<lcomp>
<head>nesting start</head>
<incomp>
<count>3</count>
<comp:gcompel>
<fstr>first</fstr>
<multel>
```

```
<in1>1</in1>
<in2>C</in2>
<in3>2.45E+1</in3>
</multel>
</comp:gcompel>
<comp:gcompel>
<fstr>second</fstr>
<multel>
<in1>1</in1>
<in2>D</in2>
<in3>7.625E+3</in3>
</multel>
</comp:gcompel>
<comp:gcompel>
<fstr>third</fstr>
<multel>
<in1>0</in1>
<in2>C</in2>
<in3>4.9E+0</in3>
</multel>
</comp:gcompel>
</incomp>
<footer>nesting end</footer>
</lcomp>
<repstr>abc</repstr>
<repstr>def</repstr>
<repstr>ghi</repstr>
<repstr>jkl</repstr>
<repstr>mno</repstr>
</comp:addev3>
```

```
<comp:addev3 xmlns:comp="http://www.complex.net">
<intrep>45</intrep>
<intrep>12</intrep>
<intrep>920</intrep>
<comp2>
<bool1>1</bool1>
<dat2>2005-06-24</dat2>
</comp2>
<binel><![CDATA[3132333435]]></binel>
<lelem latt="24">twenty four</lelem>
<lcomp>
<head>nesting start</head>
<incomp>
<count>5</count>
<comp:gcompel>
<fstr>first</fstr>
<multel>
<in1>1</in1>
<in2>C</in2>
<in3>2.45E+1</in3>
</multel>
</comp:gcompel>
<comp:gcompel>
<fstr>second</fstr>
<multel>
<in1>1</in1>
<in2>D</in2>
<in3>7.625E+3</in3>
</multel>
</comp:gcompel>
<comp:gcompel>
<fstr>third</fstr>
<multel>
<in1>0</in1>
<in2>C</in2>
<in3>4.9E+0</in3>
</multel>
```

```
</comp:gcompel>
<comp:gcompel>
<fstr>fourth</fstr>
<multel>
<in1>1</in1>
<in2>F</in2>
<in3>2.98E+1</in3>
</multel>
</comp:gcompel>
<comp:gcompel>
<fstr>fifth</fstr>
<multel>
<in1>0</in1>
<in2>D</in2>
<in3>8.57E-2</in3>
</multel>
</comp:gcompel>
</incomp>
<footer>nesting end</footer>
</lcomp>
<repstr>abc</repstr>
</comp:addev3>
```

```
<comp:addev3 xmlns:comp="http://www.complex.net">
<dur>P2Y2M</dur>
<comp3>
<int1>6</int1>
<dec1>2821.54</dec1>
</comp3>
<comp3>
<int1>41</int1>
<dec1>0.02</dec1>
</comp3>
<binel><![CDATA[3132333435]]></binel>
<lelem latt="24">twenty four</lelem>
<lcomp>
<head>nesting start</head>
<incomp>
<count>0</count>
</incomp>
<footer>nesting end</footer>
</lcomp>
<repstr>abc</repstr>
<repstr>def</repstr>
<repstr>ghi</repstr>
<repstr>jkl</repstr>
<repstr>mno</repstr>
<repstr>pqr</repstr>
<repstr>stu</repstr>
<repstr>vwx</repstr>
</comp:addev3>
```

You have created the following resources:

- A message called addev3 in the message definition COMPLEX
- A message flow called addev3, which contains the mapping file `addev3_Mapping.msgmap`
- Files that contain instance messages for test

Now deploy the message set and message flow.

Deploy the message set and message flow:

This step is the sixth task in creating "Scenario D: Complex message enrichment" on page 623. Learn how to deploy the message set and message flow and run the instance messages through the broker.

1. Create a BAR file called addev3.
2. Add the message set MAPPING3_COMPLEX_messages and the message flow addev3 to the BAR file.
3. Deploy the BAR file to the broker.
4. Put the instance documents on the input queue.

The output messages look like this:

```
<comp:addev3 xmlns:comp="http://www.complex.net">
<sstr>first</sstr>
<comp1>
<dat1>2005-06-24</dat1>
<sval>date value</sval>
</comp1>
<binel><![CDATA[3132333435]]></binel>
<lelem latt="24">twenty four</lelem>
<lcomp>
<head>nesting start</head>
<incomp>
<count>3</count>
<comp:gcompel>
<fstr>first</fstr>
<multel>
<in1>1</in1>
<in2>C</in2>
<in3>2.45E+1</in3>
</multel>
</comp:gcompel>
<comp:gcompel>
<fstr>second</fstr>
<multel>
<in1>1</in1>
<in2>D</in2>
<in3>7.625E+3</in3>
</multel>
</comp:gcompel>
<comp:gcompel>
<fstr>third</fstr>
<multel>
<in1>0</in1>
<in2>C</in2>
<in3>4.9E+0</in3>
</multel>
</comp:gcompel>
</incomp>
<footer>nesting end</footer>
</lcomp>
<repstr>abc</repstr>
<repstr>def</repstr>
<repstr>ghi</repstr>
<repstr>jkl</repstr>
<repstr>mno</repstr>
</comp:addev3>
```

```
<comp:addev3 xmlns:comp="http://www.complex.net">
<intrep>45</intrep>
<intrep>12</intrep>
<intrep>920</intrep>
<comp2>
<bool1>1</bool1>
<dat2>2005-06-24</dat2>
</comp2>
```

```
<binel><![CDATA[3132333435]]></binel>
<lelem latt="24">twenty four</lelem>
<lcomp>
<head>nesting start</head>
<incomp>
<count>5</count>
<comp:gcompel>
<fstr>first</fstr>
<multel>
<in1>1</in1>
<in2>C</in2>
<in3>2.45E+1</in3>
</multel>
</comp:gcompel>
<comp:gcompel>
<fstr>second</fstr>
<multel>
<in1>1</in1>
<in2>D</in2>
<in3>7.625E+3</in3>
</multel>
</comp:gcompel>
<comp:gcompel>
<fstr>third</fstr>
<multel>
<in1>0</in1>
<in2>C</in2>
<in3>4.9E+0</in3>
</multel>
</comp:gcompel>
<comp:gcompel>
<fstr>fourth</fstr>
<multel>
<in1>1</in1>
<in2>F</in2>
<in3>2.98E+1</in3>
</multel>
</comp:gcompel>
<comp:gcompel>
<fstr>fifth</fstr>
<multel>
<in1>0</in1>
<in2>D</in2>
<in3>8.57E-2</in3>
</multel>
</comp:gcompel>
</incomp>
<footer>nesting end</footer>
</lcomp>
<repstr>abc</repstr>
</comp:addev3>
<comp:addev3 xmlns:comp="http://www.complex.net">
<dur>P2Y2M</dur>
<comp3>
<int1>6</int1>
<dec1>2821.54</dec1>
</comp3>
<comp3>
<int1>41</int1>
<dec1>0.02</dec1>
</comp3>
<binel><![CDATA[3132333435]]></binel>
<lelem latt="24">twenty four</lelem>
<lcomp>
<head>nesting start</head>
<incomp>
<count>0</count>
```

```
</incomp>
<footer>nesting end</footer>
</lcomp>
<repstr>abc</repstr>
<repstr>def</repstr>
<repstr>ghi</repstr>
<repstr>jkl</repstr>
<repstr>mno</repstr>
<repstr>pqr</repstr>
<repstr>stu</repstr>
<repstr>vwx</repstr>
</comp:addev3>
```

Next: Continue with the next step, "Develop a message flow for populating a target from a database."

Develop a message flow for populating a target from a database:

Develop a message flow that updates a message from a database for mapping scenario D, complex message enrichment.

Before you start

Perform the previous tasks for this scenario, described in the following steps:

1. "Develop a message flow that contains other nodes" on page 623
2. "Develop a message flow to map target fields from multiple other fields" on page 626
3. "Develop a message flow and message model for mapping a complex nested, repeating message" on page 628

This step is the seventh task in creating "Scenario D: Complex message enrichment" on page 623. Develop a message flow for populating a target from a database, and create a corresponding message model and instance documents.

1. Create a database called MAPDB and create a table called TRANSACTION, which has the following columns:

   ```
   ACCOUNT        VARCHAR(12)
   TDATE               DATE
   VALUE             DECIMAL(8,2)
   ```

2. Populate the database with the values shown:

   ```
   '12345678901', '2005-04-25', -14.25
   '12345678901', '2005-04-25', 100.00
   '12345678901','2005-05-15', 2891.30
   '12345678901','2005-06-11', -215.28
   ```

3. Create a Windows ODBC Data Source Name for the database.
4. Add a definition for the database to the workbench by clicking **File** → **New** → **Database Definition File**. For information about how to complete this step, including how to choose a supported database and version, follow the instructions provided in "Adding database definitions to the workbench" on page 578.
5. In the COMPLEX message definition, in namespace www.complex.net, create a message called addev4in, which has the following structure:

   ```
   addev4in
      account            (xsd:string) local element
      tdate                (xsd:date) local element
   ```

6. In the COMP2 message definition, in namespace www.comp2.net, create a message called addev4out, which has the following structure:

```
addev4out
   account             (xsd:string) local element
   tdate                 (xsd:date) local element
   value                 (xsd:decimal) local element, minOcc=0, maxOcc=-1
```

7. Create a message flow called addev4, which contains the following structure: MQInput -> Mapping -> MQOutput.
8. Open the map for the Mapping node, and select addev4in as the source and addev4out as the target.
9. Map the input to outputs as shown:

```
account --- account
tdate --- tdate
```

10. In the Message Mapping editor Spreadsheet pane, right-click the target value, and click **Select Data Source**.
11. Select MAPDB, and click **Finish**.
12. In the top pane, expand the MAPDB tree and wire the values as shown:

```
VALUE  --- value
```

13. In the Spreadsheet pane, select the target $db:select and change fn:true() to: `$db:select.MAPDB.SCHEMA.TRANSACTION.ACCOUNT=$source/comp:addev4in/account` and `$db:select.MAPDB.SCHEMA.TRANSACTION.TDATE=$source/comp:addev4in/tdate`
14. Expand the Properties tree and set the following values:

```
MessageType     |    'addev4out'
```

15. Set the data source property for the Mapping node to MAPDB.
16. Create the following instance messages with appropriate MQRFH2 headers:

```
<comp:addev4in xmlns:comp="http://www.complex.net">
<account>12345678901</account>
<tdate>2005-05-15</tdate>
</comp:addev4in>
```

```
<comp:addev4in xmlns:comp="http://www.complex.net">
<account>12345678901</account>
<tdate>2005-04-25</tdate>
</comp:addev4in>
```

You have created the following resources:

- A message called addev4in in a message definition called COMPLEX
- A message called addev4out in a message definition called COMP
- A message flow called addev4, which contains the mapping file `addev4_Mapping.msgmap`
- Files that contain instance messages

Now deploy the message set and message flow.

Deploy the message set and message flow:

This step is the eighth task in creating "Scenario D: Complex message enrichment" on page 623. Deploy the message set and message flow and run the instance messages through the broker.

1. Create a BAR file called addev4.
2. Add the message set MAPPING3_COMPLEX_messages and the message flow addev4 to the BAR file.
3. Deploy the BAR file to the broker.
4. Put the instance documents on the input queue.

The output messages have the following content:

```
<c2:addev4out xmlns:c2="http://www.comp2.net" xmlns:comp="http://www.complex.net" >
<account>12345678901</account>
<tdate>2005-05-15</tdate>
<value>2891.3</value>
</c2:addev4out>
```

Next: Continue with the next step, "Develop a message flow using a user-defined ESQL function."

Develop a message flow using a user-defined ESQL function:

Develop a message flow that calls an ESQL function for mapping scenario D, complex message enrichment.

Before you start

Perform the previous tasks for this scenario, described in the following steps:

1. "Develop a message flow that contains other nodes" on page 623
2. "Develop a message flow to map target fields from multiple other fields" on page 626
3. "Develop a message flow and message model for mapping a complex nested, repeating message" on page 628
4. "Develop a message flow for populating a target from a database" on page 634

This step is the ninth task in creating "Scenario D: Complex message enrichment" on page 623. Develop a message flow that uses a user-defined ESQL function, and create corresponding message models and instance documents.

1. In the COMPLEX message definition, in namespace www.complex.net, create messages called addev5in and addev5out, which have the following structures:

```
addev5in
   value1                   (xsd:decimal) local element
   operator                (xsd:string) local element
   value2                   (xsd:decimal) local element
   rate                       (xsd:decimal) local element

addev5out
   grossvalue              (xsd:decimal) local element
   netvalue                  (xsd:decimal) local element
```

2. Create a message flow called addev5, which contains the following structure: MQInput -> Mapping -> MQOutput.
3. Open the map for the Mapping node, and select addev5in as the source and addev5out as the target.
4. In the MAPPING3_COMPLEX_flows project, create an ESQL file called `addev5`, and code the following functions:

```
CREATE FUNCTION calcGrossvalue(IN value1 DECIMAL, IN operator CHAR,
 IN value2 DECIMAL) RETURNS DECIMAL
          BEGIN
              DECLARE outval DECIMAL;
              CASE operator
              WHEN 'PLUS' THEN
                  SET outval = value1 + value2;
              WHEN 'MINUS' THEN
                  SET outval = value1 - value2;
              WHEN 'MULTIPLY' THEN
                  SET outval = value1 * value2;
              WHEN 'DIVIDE' THEN
                  SET outval = value1 / value2;
```

```
            ELSE
                THROW USER EXCEPTION MESSAGE 2949 VALUES('Invalid Operator', operator);
                SET outval = -999999;
            END CASE;
          RETURN outval;
        END;
CREATE FUNCTION calcNetvalue(IN value1 DECIMAL, IN operator CHAR, IN value2 DECIMAL,
 IN rate DECIMAL) RETURNS DECIMAL
    BEGIN
        DECLARE grossvalue DECIMAL;
        SET grossvalue=calcGrossvalue(value1, operator, value2);
        RETURN (grossvalue * rate );
    END;
```

5. In the Message Mapping editor Spreadsheet pane, expand the message and select grossvalue.
6. In the Expression pane, enter the expression:

```
esql:calcGrossvalue($source/comp:addev5in/value1,
$source/comp:addev5in/operator,
$source/comp:addev5in/value2)
```

7. Select the target netvalue, and in the Expression pane, enter the following expression:

```
esql:calcNetvalue($source/comp:addev5in/value1,
$source/comp:addev5in/operator,
$source/comp:addev5in/value2,
$source/comp:addev5in/rate)
```

8. Expand the Properties tree and set the following values:

```
MessageType      |       'addev5out'
```

9. Create the following instance messages, with appropriate MQRFH2 headers:

```
<comp:addev5in xmlns:comp="http://www.complex.net">
<value1>125.32</value1>
<operator>PLUS</operator>
<value2>25.86</value2>
<rate>0.60</rate>
</comp:addev5in>
```

```
<comp:addev5in xmlns:comp="http://www.complex.net">
<value1>118.00</value1>
<operator>MINUS</operator>
<value2>245.01</value2>
<rate>0.30</rate>
</comp:addev5in>
```

```
<comp:addev5in xmlns:comp="http://www.complex.net">
<value1>254.02</value1>
<operator>MULTIPLY</operator>
<value2>3.21</value2>
<rate>0.75</rate>
</comp:addev5in>
```

```
<comp:addev5in xmlns:comp="http://www.complex.net">
<value1>1456.33</value1>
<operator>DIVIDE</operator>
<value2>18.58</value2>
<rate>0.92</rate>
</comp:addev5in>
```

```
<comp:addev5in xmlns:comp="http://www.complex.net">
<value1>254.02</value1>
<operator>MOD</operator>
<value2>3.21</value2>
<rate>0.75</rate>
</comp:addev5in>
```

You have created the following resources:

- Messages called addev5in and addev5out in a message definition called COMPLEX
- A message flow called addev5, which contains the mapping file `addev5_Mapping.msgmap` and ESQL file `addev5.esql`
- Files that contain instance messages

Now deploy the message set and message flow.

Deploy the message set and message flow:

This step is the tenth task in creating "Scenario D: Complex message enrichment" on page 623. Deploy the message set and message flow, and run the instance messages through the broker.

1. Create a BAR file called addev5.
2. Add the message set MAPPING3_COMPLEX_messages and the message flow addev5 to the BAR file.
3. Deploy the BAR file to the broker.
4. Put the instance documents on the input queue.

The output messages have the following content:

```
<comp:addev5out xmlns:comp="http://www.complex.net">
<grossvalue>151.18</grossvalue>
<netvalue>90.708</netvalue>
</comp:addev5out>
```

```
<comp:addev5out xmlns:comp="http://www.complex.net">
<grossvalue>-127.01</grossvalue>
<netvalue>-38.103</netvalue>
</comp:addev5out>
```

```
<comp:addev5out xmlns:comp="http://www.complex.net">
<grossvalue>815.4042</grossvalue>
<netvalue>611.55315</netvalue>
</comp:addev5out>
```

```
<comp:addev5out xmlns:comp="http://www.complex.net">
<grossvalue>78.381593110871905274488697524219 59</grossvalue>
<netvalue>72.111065662002152852529601722282 02</netvalue>
</comp:addev5out>
```

If you do not see any message output, look for an entry in the event log with content similar to the following entry:

```
BIP2949 ( BRK.default ) A user generated ESQL exception has been thrown. The additional
information provided with this exception is: ''Invalid Operator'' ''MOD''
'addev5.Mapping.ComIbmCompute' '%5' '%6' '%7' '%8' '%9' '%10' '%11'
```

This exception was thrown by a THROW EXCEPTION statement, and shows the normal behavior of the THROW statement; this exception is user-generated, so the user action is determined by the message flow and the type of exception that is thrown.

Next: Continue with the next step, "Develop a message flow using a user-defined Java procedure."

Develop a message flow using a user-defined Java procedure:

Develop a message flow that calls a Java procedure for mapping scenario D, complex message enrichment.

Before you start

Perform the previous tasks for this scenario, described in the following steps:

1. "Develop a message flow that contains other nodes" on page 623
2. "Develop a message flow to map target fields from multiple other fields" on page 626
3. "Develop a message flow and message model for mapping a complex nested, repeating message" on page 628
4. "Develop a message flow for populating a target from a database" on page 634
5. "Develop a message flow using a user-defined ESQL function" on page 636

This step is the eleventh task in creating "Scenario D: Complex message enrichment" on page 623. Develop a message flow using a user-defined Java procedure, and create corresponding message models and instance documents.

1. In the COMPLEX message definition, in namespace www.complex.net, create messages called addev6in and addev6out, which have the following structures:

```
addev6in
    hexdata                (xsd:hexBinary) local element

addev6out
    decval                  (xsd:decimal) local element
    fltval                     (xsd:float) local element
    intval                   (xsd:int) local element
```

2. Create a message flow called addev6, which contains the following structure: MQInput -> Mapping -> MQOutput.
3. Open the map for the Mapping node, and select addev6in as the source and addev6out as the target.
4. In the MAPPING3_COMPLEX_flows project, create an ESQL file called `addev6` and put these functions in it:

```
CREATE PROCEDURE decFromBinary( IN hexval BLOB )
 RETURNS DECIMAL
 LANGUAGE JAVA
 EXTERNAL NAME "addev6.decFromBinary";

CREATE PROCEDURE fltFromBinary( IN hexval BLOB )
 RETURNS DECIMAL
 LANGUAGE JAVA
 EXTERNAL NAME "addev6.fltFromBinary";

CREATE PROCEDURE intFromBinary( IN hexval BLOB )
 RETURNS DECIMAL
 LANGUAGE JAVA
 EXTERNAL NAME "addev6.intFromBinary";
```

5. Create a Java source file called `addev6.java`, which has the following contents:

```
import java.lang.*;
import java.math.*;

public class addev6 {
    //
    // Return decimal element from binary string
    //
    public static BigDecimal decFromBinary( byte[] hexval) {
    // Look for element named decval
    String search = "decval";
    String snval = findElement(hexval ,search );
    // Convert the value to decimal type
    BigDecimal numval = new BigDecimal(snval);
    return numval;
    }
```

```
    //
    // Return float element from binary string
    //
    public static Double fltFromBinary( byte[] hexval) {
    // Look for element named fltval
    String search = "fltval";
    String snval = findElement(hexval ,search );
    // Convert the value to float type
    Double numval = new Double(snval);
    return numval;
    }
    //
    // Return integer element from binary string
    //
    public static Long intFromBinary( byte[] hexval) {
    // Look for element named intval
    String search = "intval";
    String snval = findElement(hexval ,search );
    // Convert the value to integer type
    Long numval = new Long(snval);
    return numval;
    }
    //
    // Locate the named element and its value in the binary data
    //
    private static String findElement( byte[] hexval, String search ) {
    // Convert bytes to string
    String hexstr = new String(hexval);
    // Fixed length label/value pairs (length=14)
    int nvals = hexstr.length() / 14;
    String numval = "";
    String[] label = new String[nvals];
    String[] value = new String[nvals];
    // Loop over number of label/value pairs
    for ( int i=0; i < nvals; i ++ ) {
        // get start position
        int st = i * 14;
        // label is length 6
        int endl = st + 6;
        // value is length 8
        int endv = endl + 8;
        // extract label and value from string
        label[i] = hexstr.substring( st, endl);
        value[i] = hexstr.substring( (endl+1), endv);
        // Check whether the current pair has the label requested
        if ( label[i].compareTo( search) == 0 ) {
        // trim padding from the value
        numval = value[i].trim();
        }
    }
    return numval;
    }
}
```

6. Compile the Java code, and add the location of the class file to the system classpath. You might have to restart Windows if you edit the CLASSPATH.
7. In the Message Mapping editor Spreadsheet pane, expand the target message and set the target decval to the value `esql:decFromBinary($source/comp:addev6in/bval)`.
8. Set the target fltval to `esql:fltFromBinary($source/comp:addev6in/bval)`.
9. Set the target intval to `esql:intFromBinary($source/comp:addev6in/bval)`.
10. Expand the Properties target and set the values shown:

    ```
    MessageType     |     'addev6out
    ```

11. Create the following instance message, with appropriate MQRFH2 headers:

```
<comp:addev6in xmlns:comp="http://www.complex.net">
<bval>
<![CDATA[64656376616c20202031342e3238666c7476616c
2020312e34452b32696e7476616c2020202020313230]]>
</bval>
</comp:addev6in>
```

You have created the following resources:

- Messages called addev6in and addev6out in a message definition called COMPLEX
- A message flow called addev6, which contains the mapping file `addev6_Mapping.msgmap` and ESQL file `addev6.esql`
- A Java source file called `addev6.java` and a compiled class file called `addev6.class` in a place where the system CLASSPATH can find it
- Files that contain instance messages

Now deploy the message set and message flow.

Deploy the message set and message flow:

This step is the final task in creating "Scenario D: Complex message enrichment" on page 623. Deploy the message set and message flow, and run the instance message through the broker.

1. Create a BAR file called addev6.
2. Add the message set MAPPING3_COMPLEX_messages and the message flow addev6 to the BAR file.
3. Deploy the BAR file to the broker.
4. Put the instance documents on the input queue.

The output message has the following content:

```
<comp:addev6out xmlns:comp="http://www.complex.net">
<decval>14.28</decval>
<fltval>1.4E+2</fltval>
<intval>120</intval>
</comp:addev6out>
```

Next: You have now completed scenario D. If you want more information about how to use maps, return to "Message mapping scenarios" on page 603, and explore another scenario.

Scenario E: Resolving a choice with alternative message data

This scenario demonstrates how to resolve a choice with alternative message data.

Before you start:

1. Create the appropriate message model, either by using the tooling or by importing the message structure files (for example, C header or XML Schema Definition files).
2. Create a message flow that has the following structure:

   ```
   MQInput > Mapping > MQOutput
   ```

The following message model is used in this example:

```
chsmess  (message)
 head   (xsd:string)
 choice  (group)
```

```
 str1  (xsd:string)
 int1  (xsd:int)
 dur1  (xsd:duration)
footer  (xsd:string)
```

1. Switch to the Broker Application Development perspective.
2. Right-click the Mapping node and click **Open Map**.
3. Accept the default project and name, and click **Next**.
4. Accept the default usage and click **Next**.
5. Clear **Based on records in a database** and click **Next**.
6. Select `chsmess` as the source message and the target message, and click **OK**.
7. In the Connection pane, open the source and target trees by clicking the addition (+) icons.
8. Open the `chsmess` tree in the Source and Target panes in the same way.
9. In both Source and Target panes, click the addition (+) icon next to the `choice` group.
10. Click `head` in the Message Mapping editor Source pane and drag it onto `head` in the Target pane. A line joins them.
11. Repeat Step 10 for each corresponding element (`str1`, `int1`, `dur1`, and `footer`.)
12. In the Map Script | Value table, open the tree by clicking the `$target` + box.
13. Open the `chsmess` tree. A set of `if-elseif` elements appears.
14. Open each if or elseif element. One if or elseif statement exists for each choice; each if or elseif statement has the condition `1=1`.
15. Click the first function (for example, for `str1`) and change it in the Edit pane so that it has the content `$source/chsmess/head='str1`. If the input element `head` has a value `str1`, the assignment `str1 <- $source/chsmess/str1` takes place.
16. Click the second function (for example, for `int1`) and change it in the Expression editor so that it has the content `$source/chsmess/head='int1'`. If the input element `head` has a value `int1`, the assignment `int1 <- $source/chsmess/int1` takes place.
17. Click the third function (for example, for `dur1`) and change it in the Expression editor so that it has the content `$source/chsmess/head='dur1'`. If the input element `head` has a value `dur1`, the assignment `dur1 <- $source/chsmess/dur1` takes place.
18. Save the mapping by clicking **File** → **Save**.

You have completed this scenario. The message model contains a choice that has been resolved by using other data in the instance message.

Scenario F: Updating the value of a message element

This scenario demonstrates how to update the value of a message element.

Before you start:

1. Create the appropriate message model, either by using the workbench, or by importing the message structure files (for example, C header or XML Schema Definition files).
2. Create a message flow that has the following structure:

   ```
   MQInput > Mapping > MQOutput
   ```

The message model used in this example has the following structure:

```
simple  (message)
  int (xsd:int)
  str (xsd:str)
```

1. Switch to the Broker Application Development perspective.
2. Right-click the Mapping node and click **Open Map**.
3. Select `simple` as the source message and the target message and click **OK**.
4. In the connection pane, open the source and target trees by clicking the addition (+) icons.
5. Open the `simple` trees on both sides in the same way.
6. Select `int` in the Message Mapping editor Source pane, and drag it onto `int` in the Target pane. A line joins them.
7. Select `str` in the Message Mapping editor Source pane and drag it onto `str` in the Target pane. A line joins them.
8. In the Map Script | Value table, open the tree by clicking the `$target` + box
9. Open the `simple` tree in the same way; both `int` and `str` have values (for example, `int $source/simple/int` and `str $source/simple/str`).
10. Select the value for `int`. The value appears in the Expression Editing pane.
11. Edit the value so that it has the content `$source/simple/int + 1` and press Enter. The value in the table is updated (the input value is incremented).
12. Select the value for `str` and edit it so that it has the content `esql:upper($source/simple/str)`, and press Enter. The value in the table is updated (the input value to is changed to uppercase).
13. Save the mapping by clicking **File → Save**.

You have completed this scenario. The input and output messages have the same structure and format, but the element values have been modified.

Defining a promoted property

When you create a message flow, you can promote properties from individual nodes in that message flow to the message flow level. Properties promoted in this way override the property values that you have set for the individual nodes.

Before you start:

Read the concept topic about promoted properties.

You can perform the following tasks related to promoting properties:
- "Promoting a property"
- "Renaming a promoted property" on page 647
- "Removing a promoted property" on page 648
- "Converging multiple properties" on page 650

Some of the properties that you can promote to the message flow are also configurable; you can modify them when you deploy the broker archive file in which you have stored the message flow to each broker. If you set values for configurable properties when you deploy a broker archive file, the values that you set override values set in the individual nodes, and those that you have promoted.

Promoting a property

You can promote a node property to the message flow level to simplify the maintenance of the message flow and its nodes, and to provide common values for multiple nodes in the flow by converging promoted properties.

Before you start:
- Create a message flow
- Read the concept topic about promoted properties

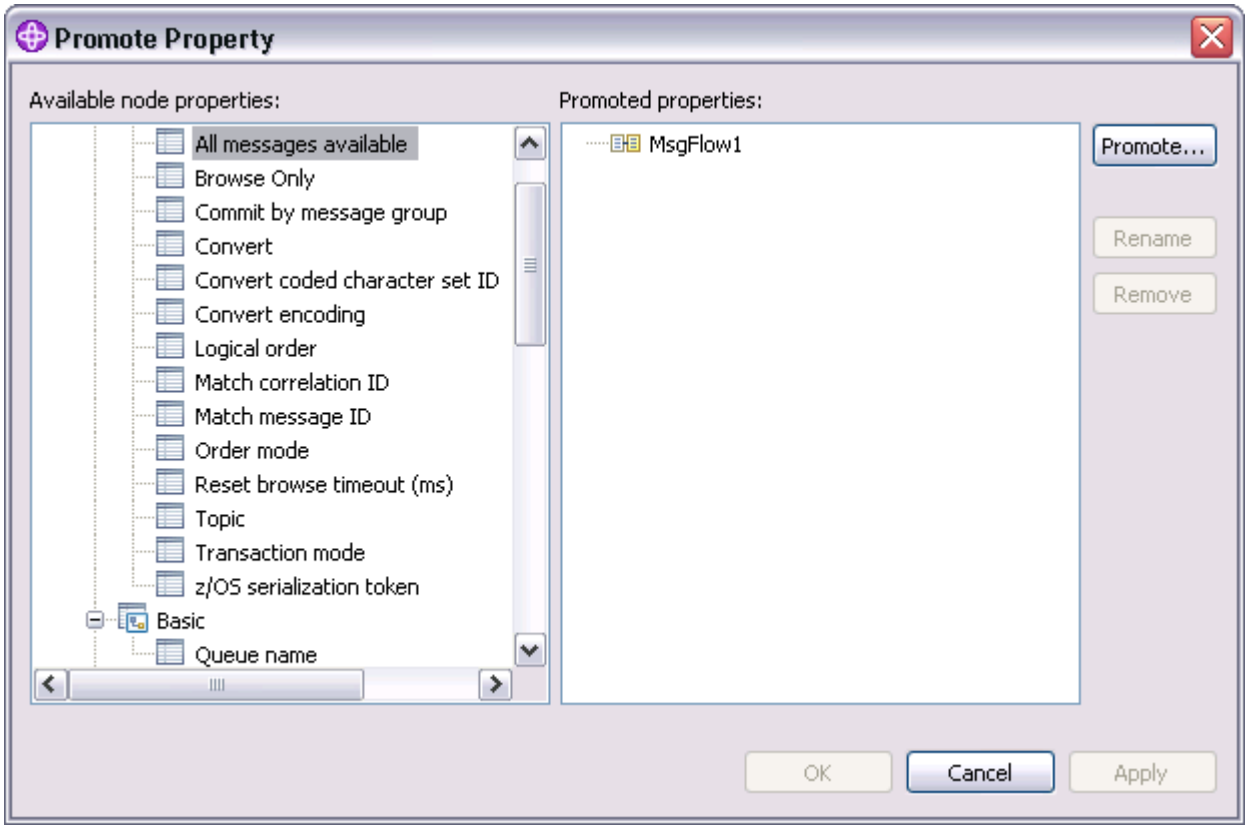
The majority of message flow node properties are available for promotion, but you cannot promote the following properties:

- Properties that name mapping modules
- A property group (but you can promote an individual property)
- A property that you cannot edit (for example, the Fix property of the MQInput node)
- The description properties (Short Description and Long Description)
- Complex properties (for example, the Query elements property of the DatabaseRoute node, or the Opaque elements property of the MQInput and several other nodes)

To promote message flow node properties to the message flow level, perform these steps.

1. Switch to the Broker Application Development perspective.
2. Open the message flow for which you want to promote properties.
3. Right-click the node whose properties you want to promote and click **Promote Property**.

   The Promote Property dialog box is displayed.

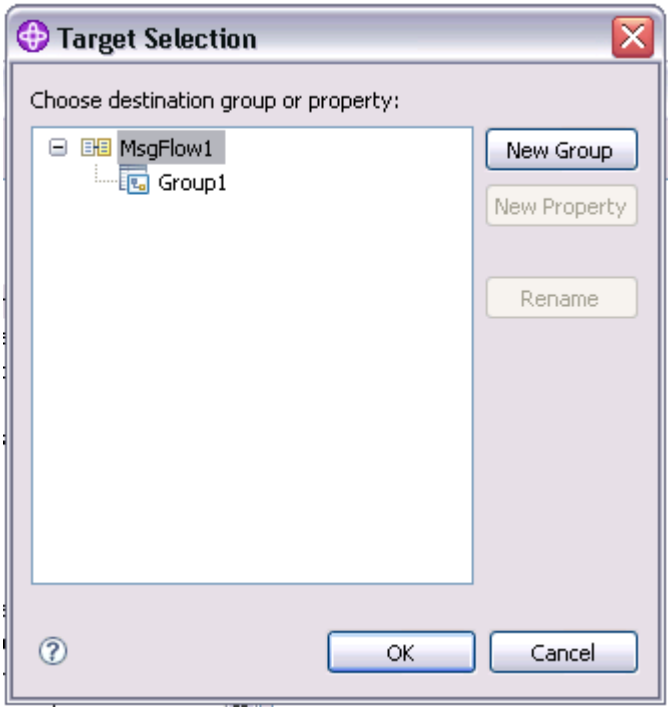


The left side of the dialog box lists all available properties for all the nodes in the message flow. The properties for the node that you clicked are expanded.

You can expand the properties for all the nodes in the open message flow, regardless of the node that you clicked initially.

The right side of the dialog box displays the name of the open message flow and all the properties that are currently promoted to the message flow. If you have not yet promoted any properties, only the message flow name is displayed as the root of the promoted property tree, as shown in the example above. If you have already promoted properties from this node, the properties appear on the right, but not on the left.

4. Select the property or properties that you want to promote to the message flow. You can select multiple properties by holding down Ctrl and selecting the properties.
5. Click **Promote**. The Target Selection dialog box opens and displays valid targets for the promotion.



6. Select the destination group or property for the properties that you want to promote. You can group together related properties from the same or different nodes in the message flow by dropping the selected properties onto a group or property that already exists, or you can create a new target for the promotion by clicking **New Group** or **New Property**. You can rename groups and properties by selecting them and clicking **Rename**.
7. Click **OK** to confirm your selections and close the Target Selection dialog box.

   If you create a new group or property using the Target Selection dialog box, the changes persist even if you select **Cancel** in the dialog box. When the dialog box closes, groups or properties that you have created using the Target Selection dialog box appear in the Promote Property dialog box. You can remove any of these properties from the Promote Property dialog box by selecting them and clicking **Remove**.
8. Click **OK** to commit your changes and close the Promoted Property dialog box. If you click **Apply**, the changes are committed but the dialog box remains open.

The message flow node properties are promoted to the message flow. When you have promoted a property, you can no longer make any changes to that property at the node level; you can update its value only at the message flow level. To view the message flow's properties, click the message flow (not the individual nodes) in the Message Flow editor to display the properties in the Properties view. The properties that you have promoted are organized in the groups that you created. If you now set a value for one of these properties, that value appears as the default value for the property whenever the message flow is included in other message flows.

When you select an embedded message flow in another message flow (a subflow) and view its properties, you see the promoted property values. If you look inside the embedded flow (by selecting **Open Subflow**), you see the original values for the properties. The value of a promoted property does not replace the original property, but it takes precedence when you deploy the message flow.

Promoting properties by dragging

You can also promote properties from the Promote Property dialog box by dragging the selected property or properties from the left pane of the Promote Property dialog box to the right pane, as described in the following steps.

1. Select the property that you want to promote. You can select multiple properties by holding down Ctrl, and selecting the properties.
2. You can promote the selected properties using the following methods:
   - Drop the selected property or properties in an empty space.

     A new group is created automatically for the message flow, and the property is placed in it, with the original name of the property and the name of the message flow node from which it came displayed beneath the property entry.

     The name of the first group that is created is Group1 by default. If a group called Group1 already exists, the group is given the name Group2, and so on. You can rename the group by double-clicking it and entering new text, or by selecting the group in the Promoted properties pane and clicking **Rename**.

     When you create a new promoted property, the name that you enter is the name by which the property is known within the system, and must meet certain Java and XML naming restrictions. These restrictions are enforced by the dialog box, and a message is displayed if you enter a name that includes a non-valid character. For example, you cannot include a space or quotation marks (").

     If you are developing a message flow in a user-defined project that will be delivered as an Eclipse plug-in, you can add translations for the promoted properties that you have added. Translated names can contain characters, such as space, that are restricted for system names. The option to provide translated strings for promoted properties is not available if you are working with a message flow in a message flow project.
   - Drop the selected property or properties onto a group that already exists, to group together related properties from the same or different nodes in the message flow.

     For example, you might want to group all promoted properties that relate to database interactions. You can change the groups to which promoted properties belong at any time by selecting a property in the Promoted properties pane and dragging it onto a different group.
   - Drop the selected property or properties onto a property that already exists, to converge related properties from the same or different nodes in the message flow.

For example, you might want to create a single promoted property that overrides the property on each node that defines a data source.

For more information on converging properties, see "Converging multiple properties" on page 650.

Promoting mandatory properties

If you promote a property that is mandatory (that is, an asterisk appears beside the name in the Properties view), the mandatory characteristic of the property is preserved. When a mandatory property is promoted, its value does not need to be set at the node level. If the flow that contains the mandatory promoted property is included as a subflow in another flow, the property must be populated for the subflow node.

Promoting properties through a hierarchy of message flows

You can repeat the process of promoting message flow node properties through several levels of message flow. You can promote properties from any level in the hierarchy to the next level above, and so on through the hierarchy to the top level. The value of a property is propagated from the highest point in the hierarchy at which it is set down to the original message flow node when the message flow is deployed to a broker. The value of that property on the original message flow node is overridden.

Renaming a promoted property

If you have promoted a property from the node to the message flow level, it is initially assigned the same name that it has at the node level. You can rename the property to have a more meaningful name in the context of the message flow.

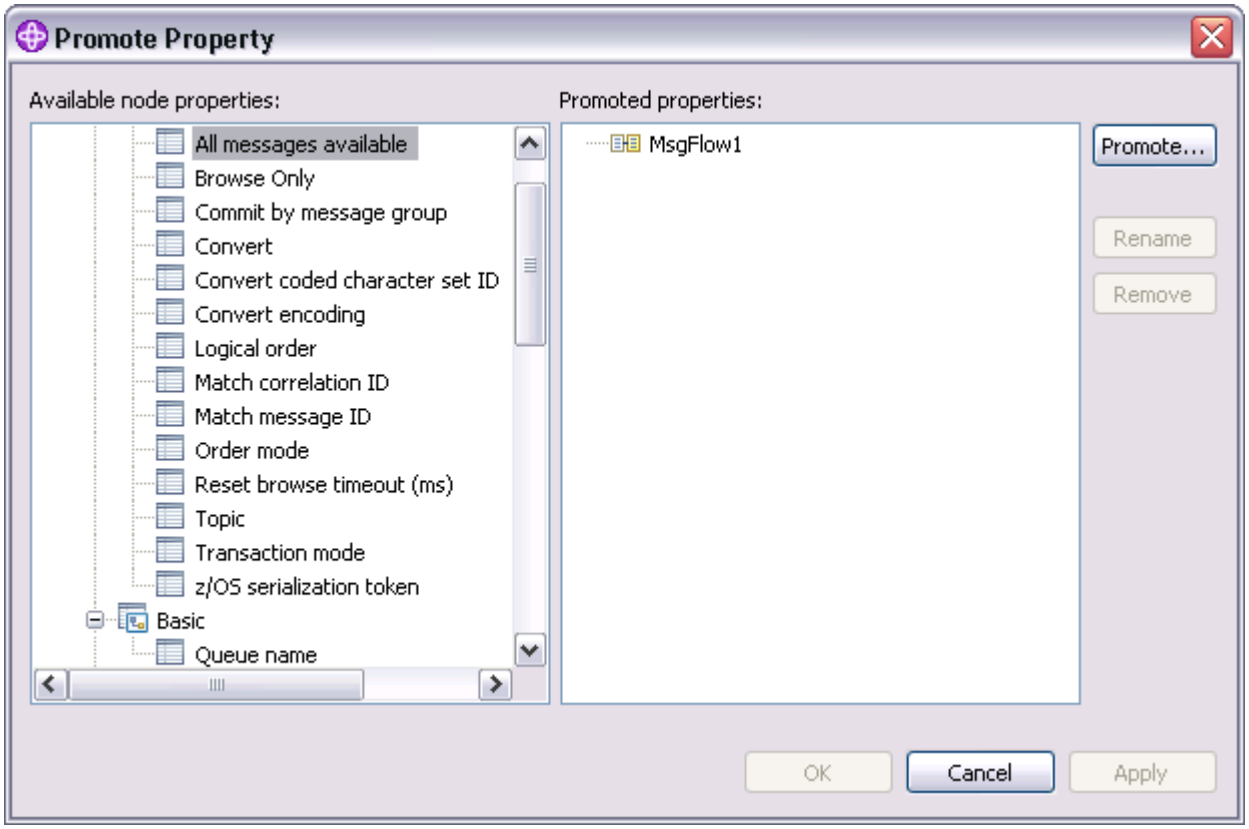
Before you start:
- Promote a property
- Read the concept topic about promoted properties

To rename a promoted property :

1. Switch to the Broker Application Development perspective.
2. Open the message flow for which you want to promote properties by double-clicking the message flow in the Broker Development view. You can also open the message flow by right-clicking it in the Broker Development view and clicking **Open** The message flow contents are displayed in the editor view.

   If this is the first message flow that you have opened, the message flow control window and the list of available built-in message flow nodes are also displayed, to the left of the editor view.
3. In the editor view, right-click the symbol of the message flow node whose properties you want to promote.
4. Select **Promote Property**.

   The Promote Property dialog is displayed.



5. Promoted properties are shown in the Promoted properties pane on the right of the Promote property dialog. Double-click the promoted property in the list of properties that are currently promoted to the message flow level, or select the property you want to rename and click **Rename**. The name is highlighted, and you can edit it. Modify the existing text or enter new text to give the property a new name, and press Enter.
6. Click **Apply** to commit this change without closing the Property Promotion dialog. Click **OK** to complete your updates and close the dialog.

Removing a promoted property

If you have promoted a property from the node to the message flow level, you can remove (delete) it if you no longer want to specify its value at the message flow level. The property reverts to the value that you specified at the node level. If you remove a promoted property that is a mandatory property, ensure that you have set a value at the node level. If you have not, you cannot successfully deploy a broker archive file that includes this message flow.

Before you start:
- Promote a property
- Read the concept topic about promoted properties

If you have promoted one or more message flow node properties, and want to delete them:

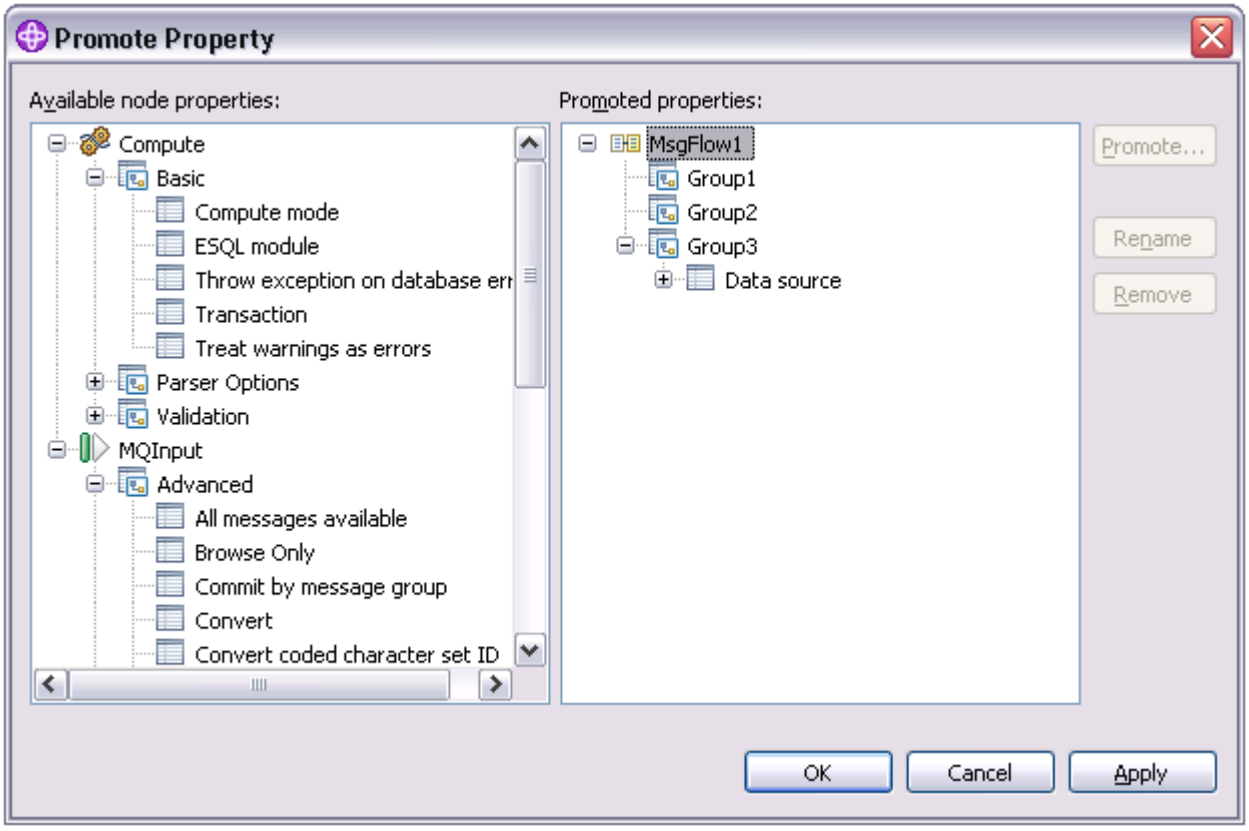
1. Switch to the Broker Application Development perspective.
2. Open the message flow for which you want to promote properties by double-clicking the message flow in the Broker Development view. You can also

open the message flow by right-clicking it in the Broker Development view and clicking **Open** The message flow contents are displayed in the editor view.

If this is the first message flow that you have opened, the message flow control window and the list of available built-in message flow nodes are also displayed, to the left of the editor view.

3. In the Editor view, right-click the symbol of the message flow node whose properties you want to promote.
4. Select **Promote Property**.

   The Promote Property dialog is displayed.



5. Select the promoted property that you want to remove in the list of properties on the right of the dialog, and click **Remove**. The property is removed from the list on the right. It is restored to the list on the left, in its appropriate place in the tree of properties for the node from which you promoted it. You can promote this property again.
6. If you want to delete all the promoted properties in a single group, select the group in the list on the right and click **Remove**. The group and all the properties it contains are deleted from this list: the individual properties that you promoted are restored to the nodes from which you promoted them.
7. Click **Apply** to commit this change without closing the Property Promotion dialog. Click **OK** to complete your updates and close the dialog.

If you have included this message flow in a higher-level message flow, and have set a value for a promoted property that you have now deleted, the embedding flow is not automatically updated to reflect the deletion. However, when you deploy that embedding message flow in the broker domain, the deleted property is ignored.

Converging multiple properties

You can promote properties from several nodes in a message flow to define a single promoted property, which applies to all those nodes.

Before you start:

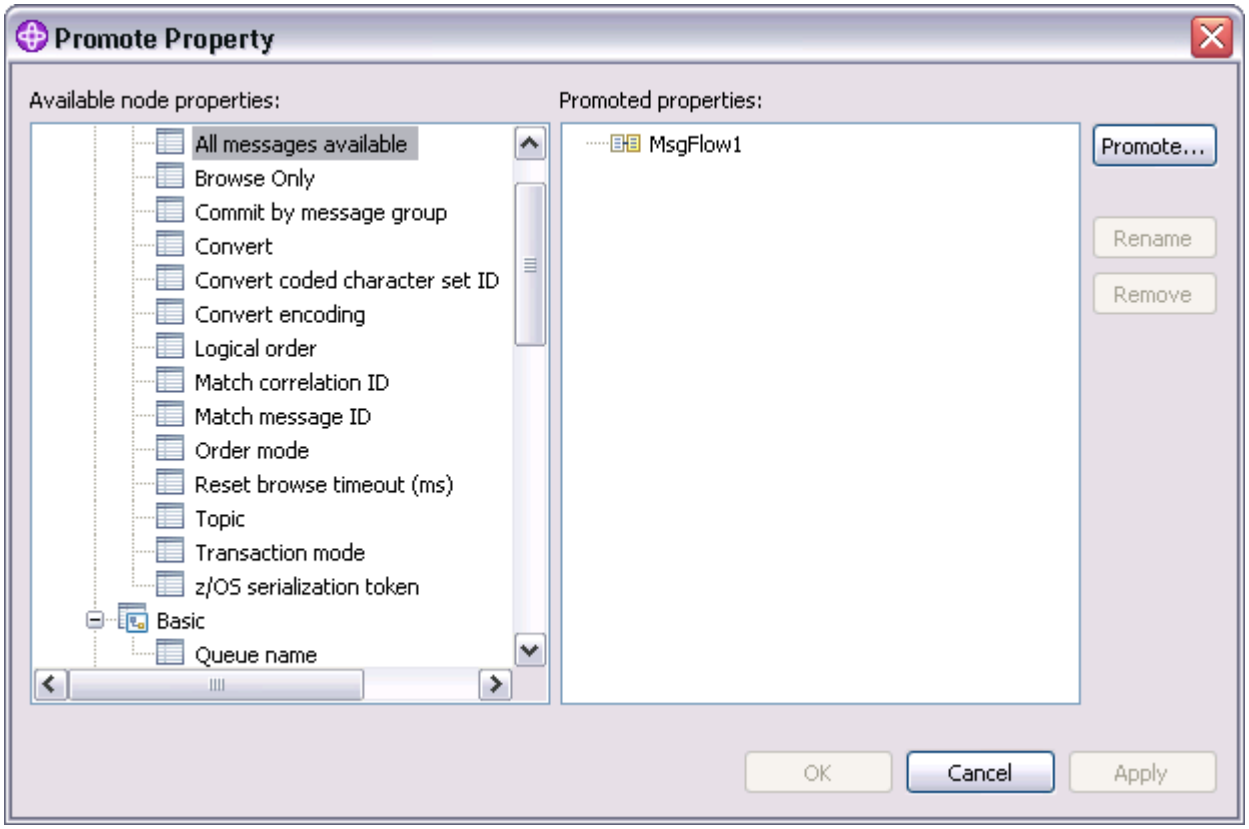
- Create a message flow
- Read the concept topic about promoted properties

One example for the use of promoting properties is for database access. If a message flow contains two Database nodes that each refer to the same physical database, you can define the physical database just once on the message flow by promoting the Data Source property of each Database node to the message flow, and setting the property at the message flow (promoted) level.

To converge multiple node properties to a single promoted property:

1. Switch to the Broker Application Development perspective.
2. Open the message flow in the Message Flow editor.
3. Right-click the node for which you want to promote the properties, then click **Promote Property**.

   The Promote Property dialog box is displayed.



4. Select the property that you want to converge. The list on the left initially shows the expanded list of all available properties for the selected node. If you have already promoted properties from this node, they do not appear on the left, but on the right.

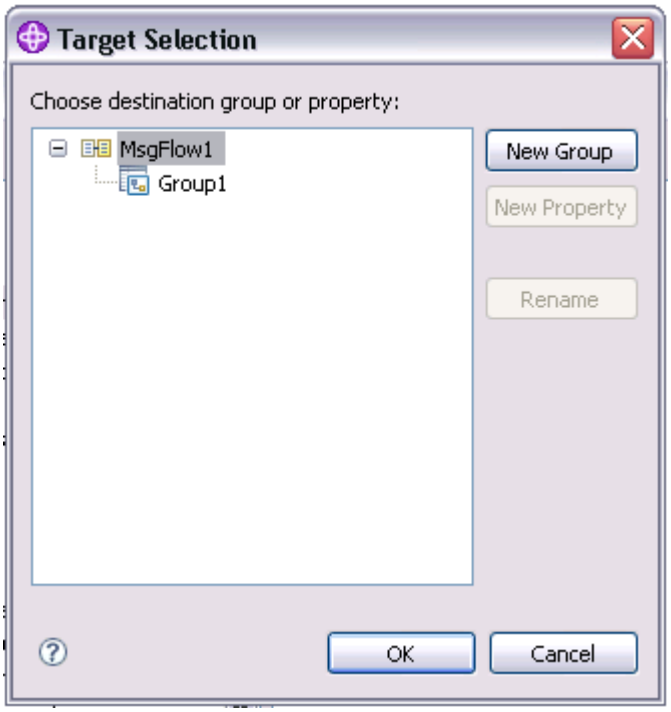
The list on the left also includes the other nodes in the open message flow. You can expand the properties that are listed under each node and work with all these properties at the same time. You do not have to close the dialog box and select another node from the Message Flow editor to continue promoting properties.

You can select multiple properties to promote by selecting a property, holding down Ctrl, and selecting one or more other properties.

If you have you selected multiple properties to converge, all the properties that you have selected must be available for promotion. If one or more of the selected properties is not available for promotion, the entire selection becomes unavailable for promotion, and the **Promote** button in the right pane is disabled.

5. Click **Promote** to promote the property or properties

   The Target Selection dialog box opens:



   The Target Selection dialog box displays only the valid targets for the promotion of the previously selected property or properties and allows you to create a new target for the promotion, such as to a new group or to a new property.

6. To converge properties from the same or different nodes in the message flow, expand the tree and click a property that already exists. You can rename the properties by selecting them and clicking **Rename**, or by double-clicking the group or property.
7. Click **OK** to confirm your selections.

   Note: If you create a new group or property by using the Target Selection dialog box, the changes persist even if you click **Cancel**. When the dialog box closes, groups or properties that you have created by using the Target Selection dialog box appear in the Promote properties dialog box.

8. Expand the property trees for all the nodes for which you want to promote properties.

9. Drag the first instance of the property that you want to converge from the list on the left, and drop it onto the appropriate group in the list on the right.
   - If the group already contains one or more promoted properties, the new property is added at the end of the group. You can rename the new property by double-clicking the property, or by selecting the property and clicking **Rename**.
   - If you want the promoted property to appear in a new group, drag the property into an empty space below the existing groups to create a new group. Alternatively:
     a. Select the property that you want to promote, and click **Promote**. The Target Selection dialog box opens.
     b. Click **New Group**, and enter the name of the new group.
     c. Click **OK** to confirm your changes.
   - If you drag the property onto an existing promoted property of a different type, a no-entry icon is displayed and you cannot drop the property. You must create this as a new promoted property, or drop it onto a compatible existing promoted property. Properties must be associated with the same property editor to be compatible. For example, if you are using built-in nodes, you can converge only properties of the same type (string with string, Boolean with Boolean).

   If you are using user-defined nodes, you must check the compatibility of the property editors for the properties that you want to converge. If you have written compiler classes for a node, you must also ensure that converged properties have the same compiler class.
10. Drag all remaining instances of the property from each of the nodes in the list on the left onto the existing promoted property. The new property is added under the existing promoted property, and is not created as a new promoted property.
11. Click **Apply** to commit this change without closing the Property Promotion dialog box. Click **OK** to complete your updates and close the dialog box.

    You can also converge properties from the Promote property dialog box by dragging the selected property or properties from the left pane of the Promote Property dialog box to the right pane:
    a. Select the property that you want to promote. You can select multiple properties to promote by selecting a property, holding down Ctrl, and selecting one or more other properties.
    b. Drop the selected property or properties onto a property in the right pane to converge related properties from the same or different nodes in the message flow.

       For example, you might want to create a single promoted property that overrides the property on each node that defines a data source.

You have promoted properties from several nodes to define a single promoted property, which is used for all those nodes.

Accessing and managing stored events

You can use configurable services to access and manage stored events.

The following types of configurable service enable you to control the queues where events are stored:
- Aggregation
- Collector

- Timer

The configurable services modify properties on the following nodes:
- AggregateControl node
- Collector node
- TimeoutNotification node

These configurable services have a **Queue prefix** property, which you can use to specify the queues in which events are stored. Different queue name prefixes can be used by different applications that are using the same type of node; as a result, you can differentiate between events that originate from different applications.

If you do not use a configurable service to set the **Queue prefix** property, the events are stored in the default queues for the type of node that you are using. The default queues are prefixed by SYSTEM.BROKER, followed by the type of node. For example, by default, the queues that are used to store the state of aggregation nodes are prefixed by SYSTEM.BROKER.AGGR,

To configure nodes to use specific storage queues for events, complete the relevant tasks:
- "Configuring the storage of events for aggregation nodes"
- "Configuring the storage of events for Collector nodes" on page 654
- "Configuring the storage of events for timeout nodes" on page 655

Configuring the storage of events for aggregation nodes

You can use an Aggregation configurable service to control the storage of events for AggregateControl and AggregateReply nodes.

By default, the storage queues used by all aggregation nodes are:
- SYSTEM.BROKER.AGGR.CONTROL
- SYSTEM.BROKER.AGGR.REPLY
- SYSTEM.BROKER.AGGR.REQUEST
- SYSTEM.BROKER.AGGR.UNKNOWN
- SYSTEM.BROKER.AGGR.TIMEOUT

However, you can control the queues that are used by different aggregation nodes by creating alternative queues containing a *QueuePrefix*, and using an Aggregation configurable service to specify the names of those queues for storing events.

Follow these steps to specify the queues that are used to store event state, and to set the expiry time of an aggregation:

1. Create the storage queues that will be used by the aggregation nodes. The following queues are required:
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.CONTROL
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.REPLY
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.REQUEST
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.UNKNOWN
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.TIMEOUT

   The *QueuePrefix* can contain any characters that are valid in a WebSphere MQ queue name, but must be no longer than 8 characters and must not begin or end with a period (.). For example, `SET1` and `SET.1` are valid queue prefixes, but `.SET1` and `SET1.` are invalid.

If you do not create the storage queues, WebSphere Message Broker creates the set of queues (based on the default queues) when the node is deployed. If the queues cannot be created, the message flow is not deployed.

2. Use the mqsicreateconfigurableservice command to create an Aggregation configurable service. You can create a configurable service to be used with either a specific aggregation or with all aggregations in an execution group.
   a. If the configurable service is to be used with a specific aggregation, create the configurable service with the same name as the Aggregate name property on the AggregateControl and AggregateReply nodes. If the configurable service is to be used with all aggregations in an execution group, create the configurable service with the same name as the execution group.
   b. Set the **Queue prefix** property to the required value.
   c. Optionally, set the **Timeout** property to control the expiry time of an aggregation.

   For example, create a configurable service called myAggregation, which specifies queues prefixed with SYSTEM.BROKER.AGGR.SET1 and a timeout of 60 seconds:

   ```
   mqsicreateconfigurableservice WBRK6_DEFAULT_BROKER -c Aggregation -o myAggregation
   -n queuePrefix,timeoutSeconds -v SET1,60
   ```

   You can use the mqsideleteconfigurableservice command to delete the Aggregation configurable service. However, the storage queues are not deleted automatically when the configurable service is deleted, so you must delete them separately.For more information, see Configurable services properties
3. In the AggregateControl and AggregateReply nodes:
   a. Ensure that the name of the Aggregation configurable service is the same as the name specified in the Aggregate name property on the **Basic** tab; for example, `myAggregation`. If no Aggregation configurable service exists with the same name as the Aggregate name, and if a configurable service exists with the same name as the execution group, that configurable service is used instead.
   b. Optional: Use the mqsichangeproperties and mqsireportproperties commands to change or view the properties of the configurable service.

The properties for the configurable service are not used by the broker until you restart or redeploy the message flow, or restart the broker.

Configuring the storage of events for Collector nodes

You can use a Collector configurable service to control the storage of events for Collector nodes.

By default, the storage queues used by all Collector nodes are:

- SYSTEM.BROKER.EDA.EVENTS
- SYSTEM.BROKER.EDA.COLLECTIONS

These queues are also used by the Resequence node.

However, you can control the queues that are used by different Collector nodes by creating alternative queues containing a *QueuePrefix*, and using a Collector configurable service to specify the names of those queues for storing events.

Follow these steps to specify the queues that are used to store event state, and to set the expiry for the collection:

1. Create the storage queues that will be used by the Collector node. The following queues are required:
   - SYSTEM.BROKER.EDA.*QueuePrefix*.EVENTS
   - SYSTEM.BROKER.EDA.*QueuePrefix*.COLLECTIONS

   The *QueuePrefix* can contain any characters that are valid in a WebSphere MQ queue name, but must be no longer than 8 characters and must not begin or end with a period (.). For example, `SET1` and `SET.1` are valid queue prefixes, but `.SET1` and `SET1.` are invalid.

   If you do not create the storage queues, WebSphere Message Broker creates the set of queues (based on the default queues) when the node is deployed. If the queues cannot be created, the message flow is not deployed.
2. Use the mqsicreateconfigurableservice command to create a Collector configurable service. You can create a configurable service to be used with either a specific collection or with all collections in an execution group.
   a. If you are creating a configurable service to be used with a specific collection, ensure that the name of the configurable service is the same as the name that you specify in the Configurable service property on the Collector node (see step 3). If you are creating a configurable service to be used with all collections in the execution group, ensure that the configurable service has the same name as the execution group.
   b. Set the **Queue prefix** property to the required value.
   c. Optionally, set the **Collection expiry** property.

   For example, create a Collector configurable service called `myCollectorService`, which uses queues prefixed with SYSTEM.BROKER.EDA.SET1, and with a collection expiry of 60 seconds:

   ```
   mqsicreateconfigurableservice WBRK6_DEFAULT_BROKER -c Collector -o myCollectorService
   -n queuePrefix,collectionExpirySeconds -v SET1,60
   ```

   You can use the mqsideleteconfigurableservice command to delete the Collector configurable service. However, the storage queues are not deleted automatically when the configurable service is deleted, so you must delete them separately.

   For more information, see Configurable services properties
3. In the Collector node:
   a. If the configurable service is to be used for a specific collection, specify the name of the configurable service in the Configurable service property on the **Advanced** tab; for example, `myCollectorService`. If you do not set the Configurable service property, and if a configurable service exists with the same name as the execution group, that configurable service is used.
   b. Optional: Use the mqsichangeproperties and mqsireportproperties commands to change or view the properties of the configurable service.

The properties for the configurable service are not used by the broker until you restart or redeploy the message flow, or restart the broker.

Configuring the storage of events for timeout nodes

You can use a Timer configurable service to control the storage of events for TimeoutNotification and TimeoutControl nodes.

By default, the storage queue used by all timeout nodes is the SYSTEM.BROKER.TIMEOUT.QUEUE

However, you can control the queues that are used by different timeout nodes by creating alternative queues containing a *QueuePrefix*, and using a Timer configurable service to specify the names of those queues for storing events.

Follow these steps to specify the queue that is used to store event state:

1. Create the storage queue that will be used by the timeout nodes. The following queue is required:
   - SYSTEM.BROKER.TIMEOUT.*QueuePrefix*.QUEUE

   The *QueuePrefix* can contain any characters that are valid in a WebSphere MQ queue name, but must be no longer than 8 characters and must not begin or end with a period (.). For example, `SET1` and `SET.1` are valid queue prefixes, but `.SET1` and `SET1.` are invalid.

   If you do not create the storage queue, WebSphere Message Broker creates the queue (based on the default queue) when the node is deployed. If the queue cannot be created, the message flow is not deployed.
2. Use the mqsicreateconfigurableservice command to create a Timer configurable service. You can create a configurable service to be used with either specific timeout requests or with all timeout requests in an execution group.
   a. If the configurable service is to be used with specific timeout requests, create the configurable service with the same name as the Unique identifier property on the TimeoutNotification and TimeoutControl nodes. If the configurable service is to be used with all timeout requests in an execution group, create the configurable service with the same name as the execution group.
   b. Set the **Queue prefix** property to the required value.

   For example, create a Timer configurable service that uses a queue prefixed with SYSTEM.BROKER.TIMEOUT.SET1:

   ```
   mqsicreateconfigurableservice WBRK6_DEFAULT_BROKER -c Timer -o myTimer
   -n queuePrefix -v SET1
   ```

   You can use the mqsideleteconfigurableservice command to delete the Timer configurable service. However, the storage queue is not deleted automatically when the configurable service is deleted, so you must delete it separately.For more information, see Configurable services properties.
3. In the TimeoutNotification and TimeoutControl nodes:
   a. Ensure that the name of the Timer configurable service is the same as the name specified in the Unique Identifier property on the **Basic** tab; for example, `myTimer`. If no Timer configurable service exists with the same name as the Unique Identifier, and if a configurable service exists with the same name as the execution group, that configurable service is used instead.
   b. Optional: Use the mqsichangeproperties and mqsireportproperties commands to change or view the properties of the configurable service.

The properties for the configurable service are not used by the broker until you restart or redeploy the message flow, or restart the broker.

Collecting message flow accounting and statistics data

You can configure your message flow to collect statistics about the operation and behavior of your message flows. Use this information to assess the performance of your message flows.

Before you start:

Read the concept topic about message flow accounting and statistics data.

The following topics describe the tasks that you can complete to control collection of message flow accounting and statistics data by using the command line:

- "Starting to collect message flow accounting and statistics data"
- "Stopping message flow accounting and statistics data collection" on page 660
- "Viewing message flow accounting and statistics data collection parameters" on page 661
- "Modifying message flow accounting and statistics data collection parameters" on page 661
- "Resetting message flow accounting and statistics archive data" on page 662

The topics listed here show examples of how to issue the accounting and statistics commands. The examples for z/OS are shown for SDSF; if you are using another interface, you must modify the example shown according to the requirements of that interface. For details of other z/OS options, see Issuing commands to the z/OS console.

Starting to collect message flow accounting and statistics data

You can start collecting message flow accounting and statistics data for an active broker at any time. You can specify what kind of statistics you want to collect, and where to send the data.

Before you start:

- Create a message flow
- Deploy a broker archive file
- Read the concept topic about message flow accounting and statistics collection options

Select the granularity of the data that you want to be collected by specifying the appropriate parameters on the mqsichangeflowstats command. You must request statistics collection on a broker basis. If you want to collect information for more than one broker, you must issue the corresponding number of commands.

To start collecting message flow accounting and statistics data:

1. Identify the broker for which you want to collect statistics .
2. Decide the resource for which you want to collect statistics. You can collect statistics for a specific execution group, or for all execution groups for the specified broker.
   - If you indicate a specific execution group, you can request that data is recorded for a specific message flow or all message flows in that group.
   - If you specify all execution groups, you must specify all message flows.
3. Decide if you want to collect thread related statistics.
4. Decide if you want to collect node related statistics. If you do, you can also collect information about terminals for the nodes.
5. Decide if you want to associate data collection with a particular accounting origin. This option is valid for snapshot and archive data, and for message flows and execution groups. However, when active, you must set its origin value in each message flow to which it refers. If you do not configure the participating message flows to set the appropriate origin identifier, the data collected for that message flow is collected with the origin set to `Anonymous`.

See “Setting message flow accounting and statistics accounting origin” on page 659 for further details.

6. Decide the target destination:
   - User trace log. This is the default setting. The output data can be processed using mqsireadlog and mqsiformatlog.
   - XML format publication message. If you chose this as your target destination, register the following topic for the subscriber:

```
$SYS/Broker/brokerName/StatisticsAccounting/recordType/executionGroupLabel/messageFlowLabel
```

     Where, *brokerName*, *executionGroupLabel*, and *messageFlowLabel* are the broker, execution group and message flow on which you want to receive data. *recordType* is the type of data collection on which you want to receive publications (snapshot, archive, or # to receive both snapshot and archive). The value that you specify for record type is case sensitive; therefore, if you choose to receive snapshot data, set the record type to SnapShot.
   - z/OS SMF (on z/OS only)
7. Decide the type of data collection that you want:
   - Snapshot
   - Archive

   You can collect snapshot and archive data at the same time, but you have to configure them separately.
8. Issue the mqsichangeflowstats command with the appropriate parameters to reflect the decisions that you have made.

   For example, to turn on snapshot data for all message flows in the default execution group for BrokerA, and include node data with the basic message flow statistics, enter:

```
mqsichangeflowstats BrokerA -s -e default -j -c active -n basic
```

   z/OS Using SDSF on z/OS, enter:

```
/F BrokerA,cs s=yes,e=default,j=yes,c=active,n=basic
```

   Refer to the mqsichangeflowstats command for further examples.

When the command completes successfully, data collection for the specified resources is started:

- If you have requested snapshot data, information is collected for approximately 20 seconds, and the results are written to the specified output.
- If you have requested archive data, information is collected for the interval defined for the broker (on the mqsicreatebroker or mqsichangebroker command, or by the external timer facility ENF). The results are written to the specified output, the interval is reset, and data collection starts again.

Next:

You can now perform the following tasks:

- “Setting message flow accounting and statistics accounting origin” on page 659
- “Stopping message flow accounting and statistics data collection” on page 660
- “Viewing message flow accounting and statistics data collection parameters” on page 661
- “Modifying message flow accounting and statistics data collection parameters” on page 661
- “Resetting message flow accounting and statistics archive data” on page 662

Setting message flow accounting and statistics accounting origin

When you request accounting origin support for collecting message flow accounting and statistics data on the mqsichangeflowstats command, you must also configure your message flows to provide the correct identification values that indicate what the data is associated with.

Before you start:

- Create a message flow
- Read the concept topic about message flow accounting and statistics accounting origin

Accounting and statistics data is associated with an accounting origin.

You can set a different value for every message flow for which data collection is active, or the same value for a group of message flows (for example, those in a single execution group, or associated with a particular client, department, or application suite).

The accounting origin setting is not used until you deploy the message flow or flows to the brokers on which they are to run. You can activate data collection, or modify it to request accounting origin support, before or after you deploy the message flow. You do not have to stop collecting data when you deploy a message flow that changes accounting origin.

To configure a message flow to specify a particular accounting origin:

1. Switch to the Broker Application Development perspective.
2. Open the message flow that you want to work with.
3. Click **Selection** above the palette of nodes.
4. Right-click a Compute, Database, or Filter node in the editor view, and click **Open ESQL**. The associated ESQL file is opened in the editor view, and your cursor is positioned at the start of the correct module. You can include the required ESQL in any of these nodes, so decide which node in each message flow is the most appropriate for this action.

   If you want to take advantage of the accounting origin support, you must include one of these nodes in each message flow for which you want a specific origin set. If you have not configured one of these three nodes in the message flow, you must add one at a suitable point (for example, immediately following the input node) and connect it to other nodes in the flow.
5. Update the ESQL in the node's module to set an accounting origin. The broker uses the origin identifier that is set in the Environment tree. You must set a value in the field with correlation name `Environment.Broker.Accounting.Origin`. This field is not created automatically in the Environment tree when the message is first received in the broker. It is created only when you set it in an ESQL module associated with a node in the message flow.

   If you do not set a value in the message flow, the default value `Anonymous` is used for all output. If you set a value in more than one place in the message flow, the value that you set immediately before the message flow terminates is used in the output data.

   The code that you need to add is of the form:

   ```
   SET Environment.Broker.Accounting.Origin = "value";
   ```

You can set the identifier to a fixed value (as shown previously), or you can determine its value based on a dynamic value that is known only at run time. The value must be character data, and can be a maximum of 32 bytes. For example, you might set its value to the contents of a particular field in the message that is being processed (if you are coding ESQL for a Compute node, you must use correlation name InputBody in place of Body in the following example):

```
IF Body.DepartmentName <> NULL THEN
   SET Environment.Broker.Accounting.Origin = Body.DepartmentName;
END IF;
```

6. Save the ESQL module, and check that you have not introduced any errors.
7. Save the message flow, and check again for errors.

You are now ready to deploy the updated message flow. Accounting and statistics data records that are collected after the message flow has been deployed will include the origin identifier that you have set.

Stopping message flow accounting and statistics data collection

You can stop collecting data for message flow accounting and statistics at any time. You do not have to stop the message flow, execution group, or broker to make this change, nor do you have to redeploy the message flow.

Before you start:
- Start to collect message flow accounting and statistics data
- Read the concept topic about message flow accounting and statistics data

You can stop collecting data for message flow accounting and statistics at any time. You do not have to stop the message flow, execution group, or broker to make this change, nor do you have to redeploy the message flow.

You can modify the parameters that are currently in force for collecting message flow accounting and statistics data without stopping data collection. See "Modifying message flow accounting and statistics data collection parameters" on page 661 for further details.

To stop collecting data:

1. Check the resources for which you want to stop collecting data.

   You do not have to stop all active data collection. You can stop a subset of data collection. For example, if you started collecting statistics for all message flows in a particular execution group, you can stop doing so for a specific message flow in that execution group. Data collection for all other message flows in that execution group continues.
2. Issue the mqsichangeflowstats command with the appropriate parameters to stop collecting data for some or all resources.

   For example, to switch off snapshot data for all message flows in all execution groups for BrokerA, enter:

   ```
   mqsichangeflowstats BrokerA -s -g -j -c inactive
   ```

   z/OS Using SDSF on z/OS, enter:

   ```
   /F BrokerA,cs s=yes g=yes j=yes c=inactive
   ```

   Refer to the mqsichangeflowstats command for further examples.

When the command completes successfully, data collection for the specified resources is stopped. Any outstanding data that has been collected is written to the output destination when you issue this command, to ensure the integrity of data collection.

Viewing message flow accounting and statistics data collection parameters

You can review and check the parameters that are currently in effect for message flow accounting and statistics data collection by using the mqsireportflowstats command.

Before you start:
- Start to collect message flow accounting and statistics data
- Read the concept topic about message flow accounting and statistics data

To view message flow accounting and statistics data collection parameters:

Issue the mqsireportflowstats command with the appropriate parameters to view the parameters that are currently being used by the broker to control archive data collection or snapshot data collection.
You can view the parameters in force for a broker, an execution group, or an individual message flow.
For example, to view parameters for snapshot data for all message flows in all execution groups for BrokerA, enter:

```
mqsireportflowstats BrokerA -s -g -j
```

z/OS Using SDSF on z/OS, enter:

```
/F BrokerA,rs s=yes,g=yes,j=yes
```

Refer to the mqsireportflowstats command for further examples.

The command displays the current status, for example:

```
BIP8187I: Statistics Snapshot settings for flow MyFlow1 in execution
group default - On?: inactive,
ThreadDataLevel: basic, NodeDataLevel:  basic,
OutputFormat: usertrace, AccountingOrigin: basic
```

Next:

You can now modify the data collection parameters.

Modifying message flow accounting and statistics data collection parameters

You can modify the parameters that you have set for message flow accounting and statistics data collection. For example, you can start collecting data for a new message flow that you have deployed to an execution group for which you are already collecting data. You can modify parameters while data collection is active; you do not have to stop data collection and restart it.

Before you start:
- Start to collect message flow accounting and statistics data
- Read the concept topic about message flow accounting and statistics data

To modify message flow accounting and statistics parameters:

1. Decide which data collection parameters you want to change. You can modify the parameters that are in force for a broker, an execution group, or an individual message flow.
2. Issue the mqsichangeflowstats command with the appropriate parameters to modify the parameters that are currently being used by the broker to control archive data collection or snapshot data collection.

   For example, to modify parameters to extend snapshot data collection to a new message flow MFlow2 in execution group EG2 for BrokerA, enter:

   ```
   mqsichangeflowstats BrokerA -s -e EG2 -f MFlow2 -c active
   ```

   z/OS Using SDSF on z/OS, enter:

   ```
   /F BrokerA,cs s=yes,e=EG2,f=MFlow2,c=active
   ```

   If you want to specify an accounting origin for archive data for a particular message flow in an execution group, enter:

   ```
   mqsichangeflowstats BrokerA -a -e EG4 -f MFlowX -b basic
   ```

   z/OS Using SDSF on z/OS, enter:

   ```
   /F BrokerA,cs a=yes,e=EG4,f=MFlowX,b=basic
   ```

   Refer to the mqsichangeflowstats command for further examples.

When the command completes successfully, the new parameters that you have specified for data collection are in force. These parameters remain in force until you stop data collection or make further modifications.

Resetting message flow accounting and statistics archive data

You can reset message flow accounting and statistics archive data to purge any accounting and statistics data not yet reported for that collecting interval. This removes unwanted data. You can request this at any time; you do not have to stop data collection and restart it to perform reset. You cannot reset snapshot data.

Before you start:

- Start to collect message flow accounting and statistics data
- Read the concept topic about message flow accounting and statistics data

To reset message flow accounting and statistics archive data:

1. Identify the broker, and optionally the execution group, for which you want to reset archive data. You cannot reset archive data on a message flow basis.
2. Issue the mqsichangeflowstats command with the appropriate parameters to reset archive data.

   For example, to reset archive data for BrokerA, enter:

   ```
   mqsichangeflowstats BrokerA -a -g -j -r
   ```

   z/OS Using SDSF on z/OS, enter:

   ```
   /F BrokerA,cs a=yes,g=yes,j=yes,r=yes
   ```

When this command completes, all accounting and statistics data accumulated so far for this interval are purged and will not be included in the reports. Data collection is restarted from this point. All archive data for all flows (indicated by `-j` or `j=yes`) in all execution groups (indicated by `-g` or `g=yes`) is reset.

This command has a minimal effect on snapshot data because the accumulation interval is much shorter than for archive data. It does not effect the settings for archive or snapshot data collection that are currently in force. When the command has completed, data collection resumes according to the current settings.

You can change any other options that are currently in effect when you reset archive data, for example accounting origin settings or output type.

Developing a user exit

Develop a user exit by declaring it, implementing its behavior, then compiling it.

To develop a user exit, follow these steps.

1. Declare the user exit.

   Declare a user exit by using the bipInitializeUserExits function to specify the following properties:

   a. Name (used to register and control the active state of the exit)
   b. User context storage
   c. A function to be invoked (for one or more Event Types)

2. Implement the user exit behavior.

   When the user exit is declared, a set of functions is registered, and these functions are invoked when specific events occur. The behavior of the user exit is provided by implementing these functions. The following table lists the events and their associated functions:

Event	Function
A message is dequeued from the input source	cciInputMessageCallback
A message is propagated to the node for processing	cciPropagatedMessageCallback
A request message is sent to the output node's transport, and transport-specific destination information is written to "WrittenDestination" in the LocalEnvironment	cciOutputMessageCallback
The node completes processing	cciNodeCompletionCallback
The transaction ends	cciTransactionEventCallback

3. Your user exit code must implement the cleanup function.

   The user exit library must implement the bipTerminateUserExits function. This function is invoked as the ExecutionGroup's process is ending, and your user exit must clear up any resources allocated during the bipInitializeUserExits function.

4. Compile.

   Use your existing process for your environment to compile your user exit. The supported C compilers are shown in Optional software support. See Compiling a C user-defined extension for more details.

5. Link the compiled code to a library with the extension `.lel` that exports the bipInitializeUserExits and bipTerminateUserExits functions.

Deploying a user exit

Deploy your user exit to the broker.

Before you start:

- Write and compile the user exit code. See "Developing a user exit."
- Ensure that the exit:
  1. Is in a library that has the extension `.lel`
  2. Exports the functions bipInitializeUserExits and bipTerminateUserExits

You can set the state of the user exit dynamically to active, or inactive, on a per-message flow basis without restarting the broker.

To deploy the user exit:

1. Install the user exit code on a broker.

   The library containing the user exit code must be installed on a file system that can be accessed by the broker. For example, the file must have read and execute authority for the user ID under which the broker runs. The broker looks in the following places for libraries that contain user exits:

   - The broker property UserExitPath defines a list of directories separated by colons (semicolons on Windows). Use the –x flag on the mqsicreatebroker or mqsichangebroker command to set this property for 32-bit execution groups for each broker.

     Alternatively, you can append the directory containing the directory that holds the extension files to the environment variable MQSI_USER_EXIT_PATH associated with the environment in which the broker is running.

     If both are set, the environment variable takes precedence. All the directories in the environment variable are searched in the order in which they appear in the variable, then all the directories in the broker property are searched in the order in which they appear in the property.
   - For 64-bit extensions, you cannot use the –x parameter to modify the exit path. Append the directory containing the directory that holds the extension files to the environment variable MQSI_USER_EXIT_PATH64.

2. Load the user exit library into the broker's processes.

   When the user exit library has been installed on the broker, you must load it in one of the following ways:

   - Stop and restart the broker.
   - Run the mqsireload command to restart the execution group processes.

3. Activate the user exit.

   User exits can be active or inactive, and are inactive by default. You can change the state of a user exit dynamically by using the mqsichangeflowuserexits command on a per-flow basis, without having to restart the broker. You can also change the default state for a set of user exits to active on a per-broker basis by using the mqsichangebroker command; in this case, you do have to restart the broker.

   To set the default user exit state for a broker:

   a. Stop the broker.
   b. Set the activeUserExits property of the broker by using the mqsichangebroker command.
   c. Start the broker and check the system log to ensure that all execution groups start without error. If any invalid user exit names are specified (that is, the user exit is not provided by any library loaded by the execution group), a BIP2314 message is written to the system log and all flows in the execution groups fail to start unless you take one of the following actions:
      - Provide a library in the user exit path that implements the exit; then run the mqsireload command, or restart the broker, to load an exit from the library.
      - Run the mqsichangeflowuserexits command to remove the exit from both the active and inactive lists.

You can also override the default user exit state for a broker. You can use the mqsichangeflowuserexits command to activate, or deactivate, user exits on a per-execution group or per-message flow basis, with the order of precedence being message flow then execution group. When multiple exits are active for a flow, the broker starts them in the order that is defined by the mqsichangeflowuserexits command.

Configuring aggregation flows

Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and AggregateReply nodes.

Before you start:

Read the following concept topic:

- "Message flow aggregation" on page 161

By default, the messages are put onto a set of default storage queues (beginning SYSTEM.BROKER.AGGR) for processing, but you can use an Aggregation configurable service to specify alternative queues. You can also use the Aggregation configurable service to set a timeout for the aggregation.

For an overview of using aggregation in message flows, see "Message flow aggregation" on page 161.

To configure aggregation flows see the following topics:

- "Creating the aggregation fan-out flow"
- "Creating the aggregation fan-in flow" on page 670
- "Associating fan-out and fan-in aggregation flows" on page 674
- "Setting timeouts for aggregation" on page 676
- "Using multiple AggregateControl nodes" on page 677
- "Correlating input request and output response aggregation messages" on page 678
- "Using control messages in aggregation flows" on page 678
- "Handling exceptions in aggregation flows" on page 681
- "Configuring the storage of events for aggregation nodes" on page 653

The following sample demonstrates the use of aggregation message flows:

Airline Reservations

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

Creating the aggregation fan-out flow

The aggregation fan-out flow receives the initial input message and restructures it to present a number of requests to a number of target applications.

Before you start:

- Read the aggregation overview, see "Message flow aggregation" on page 161.
- Create a message flow project, see "Creating a message flow project" on page 256.

You can include the fan-out and fan-in flow in the same message flow. However, you might prefer to create two separate flows. For more information about the benefits of configuring separate message flows, see "Associating fan-out and fan-in aggregation flows" on page 674.

To review an example of a fan-out flow that is supplied with WebSphere Message Broker, see the following sample:

- Airline Reservations

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

To create the fan-out flow:

1. Switch to the Broker Application Development perspective.
2. Create a new message flow to provide the fan-out processing.
3. Add the following nodes in the editor view and configure and connect them as described:

   Input node

   The input node receives an input message from which multiple request messages are generated. This node can be any one of the built-in nodes, or a user-defined input node.

   a. Select the input node to open the Properties view. The node properties are displayed.
   b. Specify the source of input messages for this node. For example, specify the name of a WebSphere MQ queue in the Basic property Queue name from which the MQInput node retrieves messages.
   c. Optional: specify values for any other properties that you want to configure for this node. For example, set the Advanced property Transaction mode to the default Yes, to ensure that aggregate request messages are put under syncpoint. This option avoids the situation where the AggregateReply node receives response messages before it has received the control message that informs it of the aggregation instance. Putting the fan-out flow under transactional control ensures that the fan-out flow completes before any response messages get to the AggregateReply.
   d. Connect the input node's Out terminal to the In terminal of an AggregateControl node. This option represents the simplest configuration; if appropriate, you can include other nodes between the input node and the AggregateControl node. For example, you might want to store the request for audit purposes (in a Warehouse node), or add a unique identifier to the message (in a Compute node).
   e. Optional: if your fan-out and fan-in flows are combined in one message flow, modify the Order mode property on the Advanced tab. Select the By Queue Order option and ensure that the Logical Order property is also selected. These options force the input node to be single threaded in order to maintain the logical order of the messages that arrive on the queue. Any additional instance threads that you make available are then shared amongst only the fan-in input nodes to improve the performance of aggregation. If your fan-in and fan-out flows are in separate message flows this step is not required because you can make additional threads available specifically to the fan-in flow.

AggregateControl node

The AggregateControl node updates the LocalEnvironment associated with the input message with information required by the AggregateRequest node. The AggregateControl node creates the LocalEnvironment.ComIbmAggregateControlNode folder. This folder and the fields in it are for internal use by WebSphere Message Broker and you should not rely on their existence or values when developing your message flows.

a. Select the AggregateControl node to open the Properties view. The node properties are displayed.

b. Set the Aggregate name property of the AggregateControl node to identify this particular aggregation. It is used later to associate this AggregateControl node with a specific AggregateReply node. The Aggregate name that you specify must be contextually unique in a broker.

c. Optional: set the Timeout property to specify how long the broker waits for replies to arrive before taking some action (described in "Setting timeouts for aggregation" on page 676). If a timeout is not set on the AggregateControl node then aggregate requests stored internally will not be removed unless all aggregate reply messages return. This situation might lead to a gradual build up of messages on the internal queues. To avoid this situation, set the timeout to a value other than zero (zero means that a timeout never occurs) so that when the timeout is reached the requests are removed and the queues do not fill up with redundant requests. Even if timeouts are not required or expected, it is good practice to set the timeout value to a large value (for example: 86400 seconds, which is 24 hours) so that the queues occasionally get cleared of old aggregations.

d. Connect the Out terminal of the AggregateControl node to the In terminal of one or more Compute nodes that provide the analysis and breakdown of the request in the input message that is propagated on this terminal.

Attention: The Control terminal of the AggregateControl node was deprecated at Version 6.0 and by default any connections from this terminal to the AggregateReply node (either direct or indirect) are ignored. This configuration maximizes the efficiency of aggregation flows and does not damage the reliability of aggregations. This configuation is the optimum configuration.

However, if you do want a control message to be sent from the AggregateControl node to the AggregateReply node, you must connect the Control terminal to the corresponding AggregateReply node on the fan-in flow (either directly or indirectly, as described in "Associating fan-out and fan-in aggregation flows" on page 674). If you connect it indirectly to the AggregateReply node, for example through an MQOutput node, you must include a Compute node to add the appropriate headers to the message to ensure that it can be safely transmitted.

In addition, for the Control terminal and connections from it to be recognized, you must enable the environment variable MQSI_AGGR_COMPAT_MODE. However, choosing this option has implications regarding the performance and behavior of message aggregations. For a full description of these implications and the environment variable, see "Using control messages in aggregation flows" on page 678.

Compute node

The Compute node extracts information from the input message and constructs a new output message.

If the target applications that handle the subtask requests can extract the information that they require from the single input message, you do not need to include a Compute node to split the message. You can pass the whole input message to all target applications.

If your target applications expect to receive an individual request, not the whole input message, you must include a Compute node to generate each individual subtask output message from the input message. Configure each Compute node in the following way, copying the appropriate subset of the input message to each output message:

a. Select the Compute node to open the Properties view. The node properties are displayed.
b. Select a value for the Basic property Compute mode. This property specifies which sections of the message tree are modified by the node. The AggregateControl node inserts elements into the LocalEnvironment tree in the input message that the AggregateRequest node reads when the message reaches it. Ensure that the LocalEnvironment is copied from the input message to the output message in the Compute node. This configuration happens automatically unless you specify a value that includes LocalEnvironment (one of All, LocalEnvironment, LocalEnvironment and Message, or Exception and LocalEnvironment).

   If you specify one of these values, the broker assumes that you are customizing the Compute node with ESQL that writes to LocalEnvironment, and that you will copy over any elements in that tree that are required in the output message.

If you want to modify LocalEnvironment, add the following statement to copy the required aggregate information from input message to output message:

```
SET OutputLocalEnvironment.ComIbmAggregateControlNode =
           InputLocalEnvironment.ComIbmAggregateControlNode;
```

c. Optional: specify values for any other properties that you want to configure for this node.

d. Connect the Out terminal of each Compute node to the In terminal of the output node that represents the destination of the output request message that you have created from the input message in this node.

Output node

Include an output node for each output message that you generate in your fan-out flow. Configure each node, as described later in this section, with the appropriate modifications for each destination.

The output node must be an output node that supports the request/reply model, such as an MQOutput node, or a mixture of these nodes (depending on the requirements of the target applications).

a. Select the output node to open the Properties view. The node properties are displayed.

b. Specify the destination for the output messages for this node. For example, specify the name of a WebSphere MQ queue in the Basic property Queue name to which the MQOutput node sends messages. The target application must process its request, and send the response to the reply destination indicated in its input message (for example the WebSphere MQ ReplyToQueue).

c. Click Request in the left view and set values for these properties to specify that replies are sent to the fan-in flow's input queue.

d. Optional: specify values for any other properties that you want to configure for this node.

e. Connect the Out terminal of the output node to the In terminal of an AggregateRequest node. When the message is propagated through the output node's Out terminal, the built-in output node updates the WrittenDestination folder in the associated local environment with additional information required by the AggregateRequest node.

The information written by the built-in nodes is queue name, queue manager name, message ID and correlation ID (from the MQMD), and message reply identifier (set to the same value as message ID).

AggregateRequest node

Include an AggregateRequest node for each output message that you generate in your fan-out flow.

a. Select the AggregateRequest node to open the Properties view. The node properties are displayed.

b. Set the Basic property Folder name to a value that identifies the type of request that has been sent out. This value is used by the AggregateReply node to match up with the reply message when it is received in the fan-in flow. The folder name that you specify for each request that the fan-out flow generates must be unique.

The AggregateRequest node writes a record in WebSphere MQ for each message that it processes. This record enables the AggregateReply node to identify which request each response is associated with. If your output nodes are non-transactional, the response message might arrive at the fan-in flow before this database update is committed. For details on how you can use timeouts to avoid this situation, see "Setting timeouts for aggregation" on page 676.

CAUTION:
Although the use of timeouts can help to avoid this situation described above, configure your fan-out flow to be transactional so that response messages cannot get to the fan-in flow before the corresponding AggregateRequest nodes have committed their database updates.

4. Press Ctrl-S or click **File → Save *name*** on the taskbar menu (where *name* is the name of this message flow) to save the message flow and validate its configuration.

To collect the aggregation responses initiated by your fan-out flow create your fan-in flow, see "Creating the aggregation fan-in flow."

Creating the aggregation fan-in flow

The aggregation fan-in flow receives the responses to the request messages that are sent out by the fan-out flow, and constructs a combined response message containing all the responses received.

Before you start:

- Read an overview of aggregation in "Message flow aggregation" on page 161.
- Create a message flow project; see "Creating a message flow project" on page 256.

You can include the fan-out and fan-in flow within the same message flow. However, you might prefer to create two separate flows. For more information about the benefits of configuring separate message flows, see "Associating fan-out and fan-in aggregation flows" on page 674. Do not deploy multiple copies of the same fan-in flow either to the same or to different execution groups.

If you do not configure the fan-out flow to be transactional, the timeout values that you have specified might result in the combined response message being generated before the fan-in flow has received all the replies. For more information, see "Creating the aggregation fan-out flow" on page 665.

To review an example of a fan-in flow see the following sample:

- Airline Reservations

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

To create the fan-in flow:

1. Switch to the Broker Application Development perspective.
2. Create a message flow to provide the fan-in processing.
3. Add the following nodes in the editor view and configure and connect them as described:

Input node

The input node receives the responses to the multiple request messages that are generated from the fan-out flow.

This node must be an input node that supports the request/reply model, such as an MQInput node, or a mixture of these nodes (depending on the requirements of the applications that send these responses). The response that is received by each input node must be sent across the same protocol as the request to which it corresponds. For example, if you include an MQOutput node in the fan-out flow, the response to that request must be received by an MQInput node in this fan-in flow.

a. Select the input node to open the Properties view. The node properties are displayed.
b. Specify the source of input messages for this node. For example, specify the name of a WebSphere MQ queue in the Basic property Queue name from which the MQInput node retrieves messages.
c. Optional: specify values for any other properties that you want to configure for this node.
d. Connect the Out terminal of the input node to the In terminal of an AggregateReply node.

   Connect the terminals in this way to create the simplest configuration; if appropriate, you can include other nodes between the input node and the AggregateReply node. For example, you might want to store the replies for audit purposes (in a Warehouse node).

Include just one input node that receives all the aggregation response messages at the beginnings of the fan-in flow as described previously. If you include multiple input nodes, threads that are started by a specific reply input node might complete the aggregation and execution of the message flow while other threads are sending their response messages to the AggregateReply node and becoming eligible to timeout. Use a single input node to enable sequential processing of replies for each aggregation. Specify additional instances to provide greater processing throughput in this single node, see "Configurable message flow properties" on page 1324.

AggregateReply node

The AggregateReply node receives the inbound responses from the input node through its In terminal. The AggregateReply node stores each reply message for subsequent processing.

When all the replies for a particular group of aggregation requests have been collected, the AggregateReply node creates an aggregated reply message, and propagates it through the Out terminal.

a. Select the AggregateReply node to open the Properties view. The node properties are displayed.
b. Set the Aggregate name property of the AggregateReply node to identify this aggregation. Set this value to be the same value that you set for the Aggregate name property in the corresponding AggregateControl node in the fan-out flow.
c. Optional: to retain an unrecognized message before propagating it to the Unknown terminal, set a value for the Unknown message

timeout property. If you are using separate fan-out and fan-in flows, set this value to a non-zero number if the control message might be delayed.

d. Optional: to explicitly handle unrecognized messages, connect the Unknown terminal to another node, or sequence of nodes. If you do not connect this terminal to another node in the message flow, messages propagated through this terminal are discarded.

e. Optional: if you have specified a timeout value for this aggregation in the AggregateControl node, and you want to explicitly handle timeouts that expire before all replies are received, connect the Timeout terminal to another node, or sequence of nodes. Partially complete aggregated replies are sent to the Timeout terminal if the timer expires. If you do not connect this terminal to another node in the message flow, messages propagated through this terminal are discarded.

f. Optional: specify values for any other properties that you want to configure for this node.

g. Connect the Out terminal of the AggregateReply node to the In terminal of a Compute node.

Attention: The Control terminal of the AggregateReply node was deprecated at Version 6.0, and by default any connections to this terminal (either direct or indirect) are ignored. This change maximizes the efficiency of aggregation flows and does not damage the reliability of aggregations. This configuration provides the optimum content.

However, if you want the AggregateReply node to receive, on its Control terminal, the control message that was sent by the corresponding AggregateControl node on the fan-out flow, you must make the necessary connections as described in "Creating the aggregation fan-out flow" on page 665. Keep the path from the AggregateReply node to the output node as direct as possible to maximize the performance of aggregations. Do not modify the content of this control message.

In addition, for the Control terminal and connections to it to be recognized, you must enable the environment variable MQSI_AGGR_COMPAT_MODE. If you choose this option, the performance and behavior of message aggregations might be affected; for a full description of these implications and the environment variable, see "Using control messages in aggregation flows" on page 678.

Aggregated messages which are sent from the AggregateReply node output terminals (Out and Timeout) are not validated. Validation of data must be done before messages are sent to the AggregateReply node, because it ignores validation options when reconstructing the stored messages.

Compute node

The Compute node receives the message that contains the combined responses. Typically, the format of this combined message is not valid for output, because the aggregated reply message has an unusual structure and cannot be parsed into the bit stream required by some nodes, for example the MQOutput node. The Out and Timeout

terminals always propagate an aggregated reply message, which always requires further processing before it can be propagated to an MQOutput node. Therefore you must include a Compute node and configure this node to create a valid output message.

a. Select the Compute node to open the Properties view. The node properties are displayed.
b. Specify in the Basic property ESQL module the name of the ESQL module that customizes the function of this node.
c. Right-click the node and click **Open ESQL** to open the ESQL file that contains the module for this node. The module is highlighted in the ESQL editor view.
d. Code the ESQL to create a single output message from the aggregated replies in the input message.

   The aggregated reply message is propagated through the Out terminal. Information about how you can access its contents is provided in "Accessing the combined message contents."
e. Optional: specify values for any other properties that you want to configure for this node.
f. Connect the Out terminal of the Compute node to the In terminal of the output node that represents the destination of the single response message.

Output node

Include an output node for your fan-in flow. This node can be any of the built-in nodes, or a user-defined output node.

a. Select the output node to open the Properties view. The node properties are displayed.
b. Specify the destination for the output message for this node. For example, specify in the Basic property Queue name the name of a WebSphere MQ queue to which the MQOutput node sends messages.
c. Optional: specify values for any other properties that you want to configure for this node.

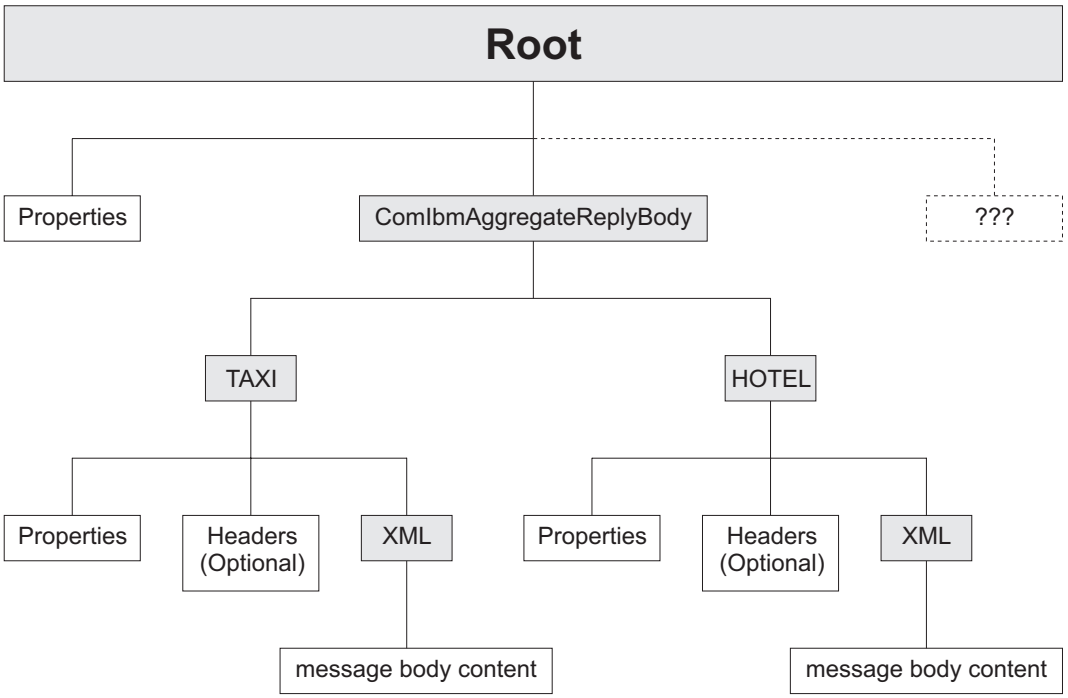
4. Press Ctrl-S or click **File** → **Save** ***name*** on the taskbar menu (where *name* is the name of this message flow) to save the message flow and validate its configuration.

Accessing the combined message contents

The AggregateReply node creates a folder in the combined message tree below Root, called ComIbmAggregateReplyBody. Below this folder, the node creates a number of subfolders using the names that you set in the AggregateRequest nodes. These subfolders are populated with the associated reply messages.

For example, the request messages might have folder names:
- TAXI
- HOTEL

The resulting aggregated reply message created by the AggregateReply node might have a structure like the following example:



Use ESQL within a Compute node to access the reply from the taxi company using the following correlation name:

```
InputRoot.ComIbmAggregateReplyBody.TAXI.xyz
```

The folder name does not have to be unique. If you have multiple requests with the folder name TAXI, you can access the separate replies using the array subscript notation, for example:

```
InputRoot.ComIbmAggregateReplyBody.TAXI[1].xyz
InputRoot.ComIbmAggregateReplyBody.TAXI[2].xyz
```

Associating fan-out and fan-in aggregation flows

Associate the fan-out message flow processing with its corresponding fan-in message flow processing by setting the Aggregate Name property of the AggregateControl and AggregateReply nodes in your aggregation flow to the same value.

If you did not configure this property when you created your fan-in and fan-out flows, you must complete this task.

Before you start:

You must have completed the following tasks:
- "Creating the aggregation fan-out flow" on page 665
- "Creating the aggregation fan-in flow" on page 670

The Aggregate Name must be contextually unique within a broker. You can have only one AggregateControl node and one AggregateReply node with a particular Aggregate Name, although you can have more than one AggregateControl node with the same Aggregate Name, see "Using multiple AggregateControl nodes" on page 677. Do not deploy a fan-in flow to multiple execution groups on the same broker; results are unpredictable.

You can either create the fan-out and fan-in flows in the same message flow, or in two different message flows. In either case, the two parts of the aggregation are associated when you set the Aggregate Name property.

How you configure your aggregation flow depends on a number of factors:

- The design of your message flow.
- The hardware on which the broker is running.
- The timeout values that you choose, see "Setting timeouts for aggregation" on page 676.
- How you expect to maintain the message flows.

You can include the fan-out and fan-in flow within the same message flow. However, you might prefer to create two separate flows. The advantages of creating separate fan-out and fan-in flows are:

- You can modify the two flows independently.
- You can start and stop the two flows independently.
- You can deploy the two flows to separate execution groups to take advantage of multiprocessor systems, or to provide data segregation for security or integrity purposes.
- You can allocate different numbers of additional threads to the two flows, as appropriate, to maintain a suitable processing ratio.

The following sample shows the use of two flows for aggregation:

- Airline Reservations

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

To associate the fan-out flow with the fan-in flow:

1. Switch to the Broker Application Development perspective.
2. Open the message flow that contains your fan-out flow.
3. Select the AggregateControl node to open the Properties view. The node properties are displayed.
4. Set the Aggregate Name property of the AggregateControl node to identify this aggregation. The Aggregate Name that you specify must be contextually unique within a broker.
5. If you have separate fan-out and fan-in flows:
   a. Press Ctrl-S or click **File** → **Save** ***name*** on the taskbar menu (where *name* is the name of this message flow) to save the message flow and validate its configuration.
   b. Open the message flow that contains your fan-in flow.
6. Select the AggregateControl node to open the Properties view. The node properties are displayed.
7. Set the Aggregate Name property of the AggregateReply node to the same value that you set for the Aggregate Name property in the corresponding AggregateControl node in the fan-out flow.
8. Press Ctrl-S or click **File** → **Save** ***name*** to save the message flow and validate its configuration.

In WebSphere Message Broker, fan-out and fan-in flows were also associated by sending control messages from the AggregateControl node to the AggregateReply

node. This facility is no longer available. For optimum performance, do not connect the AggregateControl and AggregateReply node. However, if you do want to use control messages in your aggregations, and you want to connect these two nodes, see "Using control messages in aggregation flows" on page 678.

Setting timeouts for aggregation

You can use two properties of the aggregation nodes to set timeout values for aggregated message processing.

Before you start:

To complete this task, you must have completed the following tasks:
- "Creating the aggregation fan-out flow" on page 665
- "Creating the aggregation fan-in flow" on page 670

There are two situations that might require the use of timeouts:

- In some situations you might need to receive an aggregated reply message within a specified time. Some reply messages might be slow to return, or might never arrive. For these situations:
  1. Open the fan-out message flow.
  2. Set the Timeout property of the AggregateControl node to specify how long (in seconds) the broker must wait for replies. By default, this property is set to 0, which means that there is no timeout and the broker waits indefinitely.

  If the timeout interval passes without all the replies arriving, the replies that have arrived are turned into an aggregated reply message by the corresponding AggregateReply node, and propagated to its timeout terminal. You can process this partial response message in the same way as a complete aggregated reply message. If you prefer, you can provide special processing for incomplete aggregated replies.
- When a message arrives at the in terminal of an AggregateReply node, it is examined to see if it is an expected reply message. If it is not recognized, it is propagated to the Unknown terminal. You might want the broker to wait for a specified period of time before propagating the message, because:
  - The reply message might arrive before the work performed by the AggregateRequest node has been transactionally committed. This situation can be avoided by configuring the Transaction mode property of the Input node as described in "Creating the aggregation fan-out flow" on page 665.
  - The reply message might arrive before the control message. This situation can be avoided by leaving the control terminal of the AggregateControl node unconnected. For further information about the implications of connecting the control terminal, see "Using control messages in aggregation flows" on page 678.

  These situations are most likely to happen if you send the request messages out of syncpoint, and might result in valid replies being sent to the unknown terminal. To reduce the likelihood of this event:
  1. Open the fan-in message flow.
  2. Set the Unknown Message Timeout property on the AggregateReply node. When you set this property, a message that cannot be recognized immediately as a valid reply is held persistently in the broker for the number of seconds that you specify for this property.

  If the unknown timeout interval expires, and the message is recognized, it is processed. The node also checks to see if this previously unknown message is

the last reply needed to make an aggregation complete. If it is, the aggregated reply message is constructed and propagated.

If the unknown timeout interval expires and the message is still not recognized, the message is propagated to the unknown terminal.

Using multiple AggregateControl nodes

You might find it useful to design a fan-out flow with multiple AggregateControl nodes, all with the same value set for the property *Aggregate Name*, but with different values for the *Timeout* property. This is the only situation in which you can reuse an Aggregate Name.

Before you start:

To complete this task, you must have completed the following task:
- "Creating a message flow project" on page 256

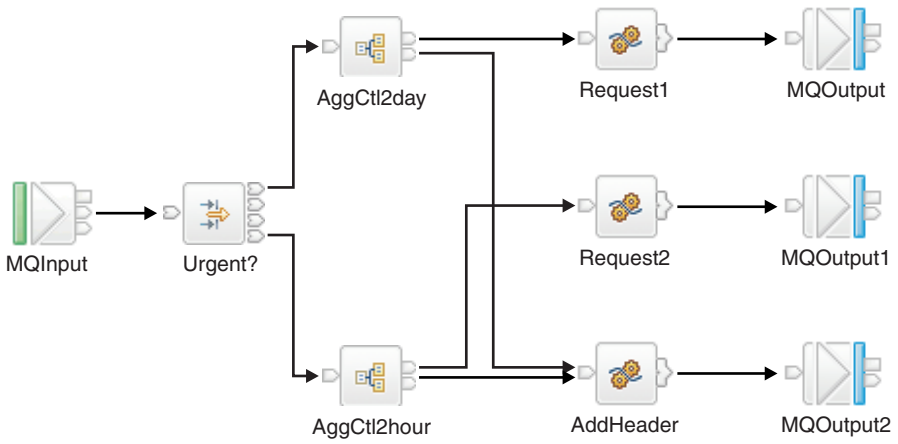
For example, if you have created an aggregation flow that books a business trip, you might have some requests that need a reply within two days, but other, more urgent requests, that need a reply within two hours.

To configure an aggregation flow that uses multiple AggregateControl nodes:

1. Switch to the Broker Application Development perspective.
2. Create or open the fan-out message flow.
3. Configure the required number of AggregateControl nodes. Set the Basic property *Aggregate Name* of each node to the same value. For example, include two nodes and enter the name `JOURNEY` as the *Aggregate Name* for both.
4. Set the value for the *Timeout* property in each node to a different value. For example, set the *Timeout* in one node to two hours; set the *Timeout* in the second node to two days.
5. Configure a Filter node to receive incoming requests, check their content, and route them to the correct AggregateControl node.
6. Connect the nodes together to achieve the required result. For example, if you have configured the Filter node to test for requests with a priority field set to urgent, connect the true terminal to the AggregateControl node with the short timeout. Connect the false terminal to the AggregateControl node with the longer timeout. Connect the out terminals of the AggregateControl nodes to the following nodes in the fan-out flow.

   You must connect the two AggregateControl nodes in parallel, not in sequence. This means that you must connect both to the Filter node (one to the true terminal, one to the false), and both to the downstream nodes that handle the requests for the fan-out. Each input message must pass through only one of the AggregateControl nodes. If you connect the nodes such that a single message is processed by more than one AggregateControl node, duplicate records are created in the database by the AggregateRequest node and subsequent processing results are unpredictable.

The following diagram shows an example fan-out message flow that uses this technique.



Correlating input request and output response aggregation messages

If you want to correlate initial request messages with their combined response messages, you can do so using the ReplyIdentifier in the Properties folder of the response message.

Before you start:

To complete this task, you must have completed the following tasks:
- "Creating the aggregation fan-out flow" on page 665
- "Creating the aggregation fan-in flow" on page 670

In some cases you might want to correlate aggregation request messages with the combined response message produced by your fan-in flow, there are two ways of doing this:

- Store some correlation information in one of the requests sent out as part of the aggregation.
- Send the original request message directly back to the AggregateReply node as one of the aggregation requests. To do this, the CorrelId must be set to the MsgId, and the MQOutput node must have its MessageContext set to 'Pass all'.

Using control messages in aggregation flows

The default behavior is that connections between AggregateControl and AggregateReply nodes for sending control messages are ignored. This configuration optimizes performance and removes the possibility that response messages will be received by the AggregateReply node before the control message.

Before you start:

To complete this task, you must have completed the following tasks:
- "Creating the aggregation fan-out flow" on page 665
- "Creating the aggregation fan-in flow" on page 670

Control messages are not required to make aggregations work correctly. However, you can send control messages in your aggregation flows if you want. To send

control messages in a message flow, see "Configuring message flows to send control messages" and "Configuring a broker environment to send control messages" on page 680.

Important: If you created message flows in Version 5.0 and configured them to use control messages, and you want to continue using control messages, see "Configuring a broker environment to send control messages" on page 680. Unless you complete this task, the connections between the AggregateControl and AggregateReply nodes that were created in earlier versions of the product are ignored.

For a working example of aggregation (without the use of control messages), see the following sample:

- Airline Reservations

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

Configuring message flows to send control messages

To configure message flows to send control messages from an AggregateControl node to an AggregateReply node:

1. Switch to the Broker Application Development perspective.
2. If you have created the fan-out and fan-in flows in a single message flow:
   a. Open the aggregation message flow.
   b. Connect the Control terminal of the AggregateControl node to the Control terminal of the AggregateReply node to make the association.

   This connection is referred to as a direct connection between the two aggregation nodes.
3. If you have created separate fan-out and fan-in message flows:
   a. Open the fan-out message flow.
   b. Configure the AggregateControl node, see "Creating the aggregation fan-out flow" on page 665.
   c. At this stage, you can configure a Compute node that creates a valid output message that contains the control message. For example, to pass the control message to an MQOutput node, configure the Compute node to add an MQMD to the message and complete the required fields in that header. For example, you can code the following ESQL:

```
SET OutputRoot.MQMD.Version = MQMD_CURRENT_VERSION;
SET OutputRoot.MQMD.Format = MQFMT_STRING;
```

   d. Configure an output node that represents the intermediate destination for the control message. For example, to send the control message to an intermediate WebSphere MQ queue, include an MQOutput node and identify the target queue in the Basic properties Queue Manager Name and Queue Name.
   e. Connect the Control terminal of the AggregateControl node to the In terminal of the Compute node, and connect the Out terminal of the Compute node to the In terminal of the output node that represents the intermediate destination for the control message.
   f. Open the fan-in message flow.

g. Configure one input node to receive the reply messages, see “Creating the aggregation fan-in flow” on page 670. This input node also receives the control information from the AggregateControl node. For example, set the Basic property Queue Name of the MQInput node to receive the response and control message from an intermediate WebSphere MQ queue.

h. Add a Filter node to your fan-in flow after the input node and before the AggregateReply node, see “Avoiding thread starvation on fan-in flows” on page 681.

i. Connect the Out terminal of the input node to the In terminal of the Filter node.

j. Connect the Out terminals of the Filter node to the Control terminal and in terminal of the AggregateReply node.

This connection is referred to as an indirect connection between the two aggregation nodes.

Configuring a broker environment to send control messages

By default, in WebSphere Message Broker Version 6.1, all connections from the Control terminal of the AggregateRequest node to the AggregateReply node are ignored. For these connections to be active, create the MQSI_AGGR_COMPAT_MODE environment variable in the broker environment. By default, the environment variable does not exist. The existence of the environment variable means that connections from the AggregateControl node are active, regardless of the value to which the environment variable is set.

When the MQSI_AGGR_COMPAT_MODE environment variable has not been created, the default behavior for aggregation fan-out flows is used. If the Control terminal of the AggregateControl node is connected, either directly or indirectly, to the In terminal of the AggregateReply node, this connection is ignored and no control message is sent.

If the MQSI_AGGR_COMPAT_MODE environment variable is created, the default behavior for aggregation fan-out flows is not used, allowing you to send control messages from the AggregateControl node to the AggregateReply node. If the Control terminal of the AggregateControl node is connected, either directly or indirectly, to the In terminal of the AggregateReply node, see “Creating the aggregation fan-out flow” on page 665, this connection is recognized and a control message is sent. Be aware that this configuration is not the optimal configuration and might affect performance.

To create the MQSI_AGGR_COMPAT_MODE variable to support connections between AggregateControl and AggregateReply nodes to be recognized:

- Windows On Windows:
  1. Open System Properties by clicking **Start** → **Control Panel** → **System**.
  2. Click the **Advanced** tab.
  3. Click **Environment Variables**.
  4. In the System variables pane, click **New**.
  5. Under **Variable name** type MQSI_AGGR_COMPAT_MODE.
  6. (Optional) You can type in the **Variable value** or leave it blank.
  7. For the environment variable to take effect, restart the computer.
- Linux UNIX z/OS On Linux, UNIX and z/OS:
  1. Edit the profile of the broker userid and include the following code:

```
export MQSI_AGGR_COMPAT_MODE=
```

2. Reload the profile.
3. Restart the broker.

Avoiding thread starvation on fan-in flows

Follow this guidance to avoid thread starvation on fan-in flows if the Control terminal of the AggregateControl node in your fan-out flow is connected to output control messages to a queue.

By not connecting the Control terminal, you can overcome the issues that are discussed here. For further information about connecting the Control terminal of the AggregateControl node, see "Using control messages in aggregation flows" on page 678.

The Aggregate Reply node has two input terminals: In and Control. The use of the Control terminal is optional. If you use both of these terminals, the MQInput nodes that supply the two terminals must not use threads from the message flow additional instance pool. If the nodes do use these threads, they compete for resources, and the Control terminal's MQInput node typically takes all available threads because it is activated before the In terminal.

Configure each MQInput node to use additional instances that are defined on the node, not at the message flow level.

Handling exceptions in aggregation flows

When you use aggregation flows, exceptions might occur.

Before you start:

Complete the following tasks:
- "Creating the aggregation fan-out flow" on page 665
- "Creating the aggregation fan-in flow" on page 670

Dealing with exceptions

If an error is detected downstream of an AggregateReply node, the broker issues an exception. Another node in the message flow might also issue an exception using the ESQL THROW statement. In either case, when an exception occurs, it is caught in one of two places:
- The input node on which the replies arrive
- The AggregateReply node

The following table lists events and describes what happens to an exception that occurs downstream of the AggregateReply node.

Event	Message propagated	Output terminal	Exception caught at
An expected reply arrives at the input node and is passed to the In terminal of the AggregateReply node. The reply is the last one that is needed to make an aggregation complete.	An aggregated reply message that contains all the replies	Out	Input node

Event	Message propagated	Output terminal	Exception caught at
An unexpected reply arrives at the input node and is passed to the AggregateReply node. The reply is not recognized as a valid reply, and the Unknown Message Timeout property is set to 0.	Message received	Unknown	Input node
A timeout occurs because all the replies for an aggregation have not yet arrived.	An aggregated reply message that contains all the replies that have been received	Timeout	AggregateReply node
An unknown timeout occurs because a retained message is not identified as a valid reply.	Retained message	Unknown	AggregateReply node
An aggregation is discovered to be complete at some time other than when the last reply arrived.	An aggregated reply message that contains all the replies	Out	AggregateReply node

To handle errors that occur in aggregation flows, you must catch these exceptions at all instances of each of these nodes in the message flow.

1. Switch to the Broker Application Development perspective.
2. Open the message flow with which you want to work.
3. To handle these exceptions yourself, connect the Catch terminal of each input and AggregateReply node to a sequence of nodes that handles the error that has occurred.

   For a unified approach to error handling, connect the Catch terminals of all these nodes to a single sequence of nodes, or create a subflow that handles errors in a single consistent manner, and connect that subflow to each Catch terminal.
4. If you want the broker to handle these exceptions using default error handling, do not connect the Catch terminals of these nodes.

If you connect the Catch terminal of the AggregateReply node, and want to send the message that is propagated through this terminal to a destination from which it can be retrieved for later processing, include a Compute node in the catch flow to provide any transport-specific processing. For example, you must add an MQMD header if you want to put the message to a WebSphere MQ queue from an MQOutput node.

The following ESQL example shows you how to add an MQMD header and pass on the replies that are received by the AggregateReply node:

```
-- Add MQMD
SET OutputRoot.MQMD.Version = 2;
.
-- Include consolidated replies in the output message
SET OutputRoot.XMLNS.Data.Parsed = InputRoot.ComIbmAggregateReplyBody;
.
```

To propagate the information about the exception in the output message, set the Compute mode property of the Compute node to a value that includes Exception.

Exceptions when dealing with unknown and timeout messages

When timeout messages or unknown messages from unknown timeout processing are produced from an AggregateReply node, they originate from an internal queue and not from an MQInput node. This behavior affects how the error handling should be performed.

If a message that is sent down the timeout thread causes an exception, the message rolls back to the AggregateReply node and is sent to the Catch terminal. If this terminal is unattached or an exception occurs while processing the message, the timeout is rolled back onto the internal queue and is reprocessed. Potentially, this behavior can lead to an infinite loop, which can be stopped by deploying a version of the message flow that fixes the problem.

To avoid this infinite loop, take the following actions.

- Connect the Catch terminal to a set of nodes that handle errors.
- Ensure that the error-handling nodes cannot throw an exception by ensuring that they perform very simple operations; for example, converting the message to a BLOB, then writing it to a queue, or adding extra TryCatch nodes.

The failure terminal of the AggregateReply node is not used currently and messages are not passed to this terminal.

Configuring the storage of events for aggregation nodes

You can use an Aggregation configurable service to control the storage of events for AggregateControl and AggregateReply nodes.

By default, the storage queues used by all aggregation nodes are:

- SYSTEM.BROKER.AGGR.CONTROL
- SYSTEM.BROKER.AGGR.REPLY
- SYSTEM.BROKER.AGGR.REQUEST
- SYSTEM.BROKER.AGGR.UNKNOWN
- SYSTEM.BROKER.AGGR.TIMEOUT

However, you can control the queues that are used by different aggregation nodes by creating alternative queues containing a *QueuePrefix*, and using an Aggregation configurable service to specify the names of those queues for storing events.

Follow these steps to specify the queues that are used to store event state, and to set the expiry time of an aggregation:

1. Create the storage queues that will be used by the aggregation nodes. The following queues are required:
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.CONTROL
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.REPLY
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.REQUEST
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.UNKNOWN
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.TIMEOUT

   The *QueuePrefix* can contain any characters that are valid in a WebSphere MQ queue name, but must be no longer than 8 characters and must not begin or end with a period (.). For example, `SET1` and `SET.1` are valid queue prefixes, but `.SET1` and `SET1.` are invalid.

If you do not create the storage queues, WebSphere Message Broker creates the set of queues (based on the default queues) when the node is deployed. If the queues cannot be created, the message flow is not deployed.

2. Use the mqsicreateconfigurableservice command to create an Aggregation configurable service. You can create a configurable service to be used with either a specific aggregation or with all aggregations in an execution group.
   a. If the configurable service is to be used with a specific aggregation, create the configurable service with the same name as the Aggregate name property on the AggregateControl and AggregateReply nodes. If the configurable service is to be used with all aggregations in an execution group, create the configurable service with the same name as the execution group.
   b. Set the **Queue prefix** property to the required value.
   c. Optionally, set the **Timeout** property to control the expiry time of an aggregation.

   For example, create a configurable service called myAggregation, which specifies queues prefixed with SYSTEM.BROKER.AGGR.SET1 and a timeout of 60 seconds:

   ```
   mqsicreateconfigurableservice WBRK6_DEFAULT_BROKER -c Aggregation -o myAggregation
   -n queuePrefix,timeoutSeconds -v SET1,60
   ```

   You can use the mqsideleteconfigurableservice command to delete the Aggregation configurable service. However, the storage queues are not deleted automatically when the configurable service is deleted, so you must delete them separately.For more information, see Configurable services properties
3. In the AggregateControl and AggregateReply nodes:
   a. Ensure that the name of the Aggregation configurable service is the same as the name specified in the Aggregate name property on the **Basic** tab; for example, `myAggregation`. If no Aggregation configurable service exists with the same name as the Aggregate name, and if a configurable service exists with the same name as the execution group, that configurable service is used instead.
   b. Optional: Use the mqsichangeproperties and mqsireportproperties commands to change or view the properties of the configurable service.

The properties for the configurable service are not used by the broker until you restart or redeploy the message flow, or restart the broker.

Creating message collections

You can use the Collector node in a message flow to generate a message collection.

Before you start:

Read the concept topic about message collections.

You can use a Collector node to group together related messages from one or more sources into a single message known as a message collection. The messages are added to the message collection based on conditions that you configure in the Collector node. You can then route the message collection to other nodes in the message flow for further processing.

You can also build a message collection manually using a Compute node or JavaCompute node.

See the following topics for instructions on configuring message flows to generate message collections:
- "Creating a flow that uses message collections"
- "Configuring the Collector node" on page 687
- "Using control messages with the Collector node" on page 694
- "Configuring the storage of events for Collector nodes" on page 654

Look at the following sample to see how to use the Collector node for message collection:

- Collector Node

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

Creating a flow that uses message collections

Use a Collector node in your message flow to group messages from one or more sources into a message collection. You can add dynamic input terminals to your Collector node for each message source that you want to configure for your message flow.

Before you start:

Read the overview "Message collections" on page 163. Complete the following task:
- "Creating a message flow project" on page 256

Look at the following sample to see how to use the Collector node for message collection:

- Collector Node

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

To create a message flow to generate and process message collections:

1. Switch to the Broker Application Development perspective.
2. Create a new message flow.
3. Add the input nodes in the editor view. The input nodes receive the messages from which message collections are generated. You can use any of the built-in nodes, or user-defined input nodes. Configure and connect them as described.
   a. Add an input node for each source of input messages for your message flow, for example an MQInput node and a JMSInput node.
   b. Select each input node in turn to display its properties in the Properties view.
   c. Specify the source of input messages for each node. For example, specify the name of a WebSphere MQ queue in the Basic property Queue Name from which the MQInput node retrieves messages.
   d. Optional: Specify values for any other properties that you want to configure for each node.
4. Add the Collector node in the editor view. The Collector node receives messages from input nodes or other nodes in the message flow. You must add a dynamic input terminal to the Collector node for each input message source before you can connect the input nodes or any upstream nodes to the Collector node. Configure and connect them as described.

a. Add a Collector node to your message flow.
b. Right-click on the Collector node and click **Add Input Terminal** to add a new dynamic input terminal to the Collector node. Add a new input terminal for each input source that you plan to add to your message flow; for more information about adding dynamic input see "Adding an input terminal to a Collector node for each input source" on page 687.
c. Connect the out terminal of each input node to a different dynamic input terminal of the Collector node. This represents the simplest configuration; if appropriate, you can include other nodes between the input node and the Collector node. For example, you might want to store the request for audit purposes (in a Warehouse node), or add a unique identifier to the message (in a Compute node).

5. Add processing nodes to your message flow. You can process message collections from a Collector node using the following nodes only:
   - Compute
   - JavaCompute

   You must connect either a Compute node or a JavaCompute node to the Collector node in your message flow. Use these nodes to process the message collection and propagate other messages. You can use ESQL or XPATH to access the contents of the individual messages in the message collection for processing. To process a message collection:
   a. Add a Compute node or a JavaCompute node to your message flow.
   b. Code your ESQL or Java to create single output messages from the message collection.
   c. Optional: Specify values for any other properties that you want to configure for this processing node.
   d. Connect the out terminal of the processing node to the in terminal of an output node or other processing node.
   e. Optional: Add other nodes to your message flow for further processing.
6. Include one or more output nodes for your message flow. These can be any of the built-in nodes, or a user-defined output node. An output node cannot process a message collection, therefore ensure that you connect the output node to a processing node that propagates single output messages. To configure an output node:
   a. Select each output node in turn to display its properties in the Properties view.
   b. Specify the destination properties for each node. For example, specify the name of a WebSphere MQ queue in the Basic property Queue Name to which the MQOutput node sends messages.
   c. Optional: Specify values for any other properties that you want to configure for each node.
7. Include processing for handling errors and expired message collections:
   a. Optional: Add processing nodes to your message flow to handle expired message collections. Connect these nodes to the Expire terminal of the Collector node.
   b. Optional: Add processing or error handling nodes to handle any exceptions in your message flow. Connect these nodes to the Catch terminal of the Collector node

   If an error is detected downstream of the Collector node, the broker throws an exception. The message collection is propagated to the Collector node's Catch terminal. Connect the Catch terminal to a sequence of nodes that handles the

errors, to avoid losing any data, and ensure that no further exceptions can be generated during error processing. The node connected to the Catch terminal must be either a Compute node or a JavaCompute node to handle the message collection.

8. Press Ctrl-S or click **File → Save *name*** on the taskbar menu (where *name* is the name of this message flow) to save the message flow and validate its configuration.

If you want to control when complete message collections are propagated, you must also add additional nodes to your message flow. For more information, see "Using control messages with the Collector node" on page 694.

Next: Configure the Collector node.

Configuring the Collector node

You can configure the Collector node to determine how messages are added to message collections. You can also use properties on the Collector node to control when message collections are propagated.

Before you start:

Complete the following tasks:
- "Creating a message flow project" on page 256
- "Creating a flow that uses message collections" on page 685

Use the following topics to configure the Collector node:
- "Adding an input terminal to a Collector node for each input source"
- "Setting event handler properties" on page 688
- "Setting the collection expiry" on page 690
- "Setting the collection name" on page 691
- "Setting the event coordination property" on page 692
- "Setting the persistence mode property" on page 693
- "Setting the configurable service property" on page 693

Adding an input terminal to a Collector node for each input source

Add new dynamic input terminals to the Collector node for all of the sources of messages for your message collections.

Before you start:

Complete the following tasks:
- "Creating a message flow project" on page 256
- "Creating a flow that uses message collections" on page 685

To add a dynamic input terminal to the Collector node for each message source:

1. Right-click the Collector node and select **Add Input Terminal**.
2. In the dialog box that is displayed, enter a name of your choice for the terminal, and click **OK**. The name that you give to the input terminal is used as the folder name in the message collection.
3. Repeat steps 1 and 2 to add further input terminals.

Next:

When you have created all the required input terminals on the Collector node, you can set the event handler properties. For more information see, "Setting event handler properties."

Setting event handler properties

You can configure event handler properties for each dynamic input terminal on a Collector node. These event handler properties determine how the messages received by each terminal are added to message collections.

Before you start:

To complete this task, you must have completed the following tasks:
- "Creating a message flow project" on page 256
- "Creating a flow that uses message collections" on page 685
- "Adding an input terminal to a Collector node for each input source" on page 687

You can use one or more of the event handler properties to control the way that messages are added to message collections for each input terminal that you added to the Collector node. Incomplete message collections are stored on a WebSphere MQ queue. The message collections are stored in the order that they are generated by the Collector node (first in, first out). Each message collection has an event handler instance for each of the input terminals. The event handler determines whether an incoming message on that terminal is added to a message collection. The event handler instance maintains information about the state of the collection, the number of messages received, the timer, and the correlation string. When a new message is received on an input terminal, it is offered to the event handler for each message collection waiting on the queue in turn. When the message is accepted by one of the event handlers, it is added to the message collection. The accepted message is not offered to any other message collections. If all the event handlers reject the message, it is added to a new message collection, which is added to the end of the queue.

The first message accepted into a collection determines the correlation string for that message collection, if it is configured. Subsequent messages offered to that message collection are only accepted if their correlation string matches that of the collection. The first message accepted by each event handler starts the timeout timer, if it is configured. Each message accepted by each event handler increments the quantity count. An event handler becomes satisfied when the number of messages accepted equals the configured quantity, or when the timeout value is reached. When an event handler is satisfied, the event handler does not accept any more messages. A message collection is complete only when all of the message collection's event handlers are satisfied. The message collection is then ready for propagation.

You can configure the event handler properties by using the **Collection Definition** table, on the **Basic** tab of the Properties view.

To configure the event handler properties on the Collector node:

1. Switch to the Broker Application Development perspective.
2. Open the message flow with the Collector node.
3. Right-click the Collector node and select **Properties**.
4. Click the **Basic** tab.
5. Use the following instructions to configure the event handler properties that you want to set for each input terminal:

- If you want to add a set number of messages to each message collection from one or more of the terminals, you must enter a value for Quantity in the Collection Definition table. This value is used to specify the number of messages that each configured input terminal accepts to complete a collection. For example, if you have set Quantity to wait for 2 messages on three of the input terminals, the message collection is not complete until 2 messages have been received on each of the three input terminals. The complete message collection contains 6 messages, 2 from each of the three terminals. As soon as more than 2 messages are received on one of the input terminals, the next message is added to a new message collection.
  a. In the Collection Definition table, click the row for the selected input terminal within the Quantity column.
  b. Enter a value for the number of input messages that you want to add to a message collection. If you select Zero or choose not to set this property, there is no limit to the number of messages accepted. In this case the value set on the Timeout property must be greater than zero. If you accept the default value of 1; only one message from the selected terminal is added to a collection.

  You must enter a value for Quantity if Timeout is not set.
- If you want to collect messages for a set amount of time before the message collection is propagated you must enter a value for Timeout. This value is used to specify the maximum time in seconds for which the selected input terminal accepts messages before completing a message collection. The timeout interval starts when the first message has arrived at the selected terminal. Any subsequent messages are added to the same message collection. When the timeout interval ends, no more messages are added to the message collection from this terminal. When the conditions on all the terminals are satisfied, the message collection is ready for propagation. When the next message reaches the selected input terminal, a new message collection is created and the timeout interval starts again. If a timeout is set on multiple input terminals, each terminal collects messages for the configured amount of time. During the timeout messages from all of the terminals are added to the same message collection.
  a. In the Collection Definition table, click the row for the selected input terminal within the Timeout column.
  b. Enter a value for the length of time in seconds that you want to wait to add messages to a message collection. For example, to wait for messages to add to a message collection for an hour, enter a value of `3600`. If you accept the default value Zero, timeout is not enabled and there is no limit on the time to wait for messages. In this case the value set on the Quantity property must be greater than zero.

  You must enter a value for Timeout if Quantity is not set.
- If you want to add messages to different message collections based on the content of the message you must enter an XPath value for the Correlation path. This value is used to specify the path in the incoming message from which to extract the correlation string. The correlation string is the value that is extracted by the correlation path. If a correlation pattern is specified, the correlation string is matched against the correlation pattern. Messages are only accepted into a message collection with the same correlation string. If you specify an asterisk (*) in the name of the message collection, it is replaced by the correlation string.
  a. In the Collection Definition table, click the row for the selected input terminal within the Correlation path column.

b. Either select a predefined correlation path from the list, or enter your own correlation path using XPath. The correlation path must begin with a correlation name, which can be followed by zero or more path fields. For example, in the following message the correlation string is `xxx` in the name field:

```
<library>
 <name>xxx</name>
 <more>
  ...
 </more>
</library>
```

In this example, the correlation path using XPath is `$Body/library/name`.

The variables `$Root`, `$LocalEnvironment`, and `$Environment` are available to allow the path to start at the roots of the message, local environment, environment trees, and message body.

If the correlation path evaluates to an empty string the unmatched message is added to a default unnamed message collection.

If you define a value for Correlation path, you can optionally configure a Correlation pattern.

- If you want to match a substring of the message content from the Correlation path, you can define a pattern to match in the message by using Correlation pattern. The Correlation pattern contains a single wildcard character and optional text. The correlation string, used for the name of the message collection, is the part of the substring that matches the wildcard. For example, if the correlation path contains the filename `part1.dat` in a file header, and the correlation pattern is specified as `*.dat`, the correlation string is `part1`.

  If this property is set, only messages that have the same correlation string are added to the same message collection. The first message added to a message collection determines the correlation string that must be matched by all other messages in that message collection.

  a. In the Collection Definition table, click the row for the selected input terminal within the Correlation pattern column.

  b. Enter a value for the correlation pattern. The Correlation pattern must contain a single wildcard character: *. This wildcard character can optionally be surrounded by other text.

  If the correlation pattern fails to match the wildcard to a substring, the unmatched message is added to a default unnamed message collection.

6. Repeat step 5 on page 688 for each of the input terminals that you added to your Collector node. You can configure different event handlers for different input sources.

Note: Ensure that you set the event handler properties across different terminals carefully to match the expected delivery of messages to the terminals on the Collector node.

You can now configure the collection expiry, see "Setting the collection expiry."

Setting the collection expiry

The collection expiry is a property on the Collector node to set a maximum timeout for adding messages to a message collection.

Before you start:

To complete this task, you must have completed the following tasks:
- "Creating a message flow project" on page 256
- "Creating a flow that uses message collections" on page 685

When messages are added to a message collection, the incomplete message collection is stored on a queue. If the message collection's event handlers are not satisfied, the incomplete message collection is stored on the queue indefinitely, and not propagated for further processing. If a Collector node has 2 input terminals, and one of the terminals stops receiving messages, for example if the source application is not running, there is the potential for the queue of incomplete message collections to grow indefinitely. To ensure that these incomplete message collections are released after an appropriate amount of time, configure the *Collection Expiry* property. You can configure this timeout, as a value in seconds, in the *Collection Expiry* property on the Collector node. The collection expiry timeout starts when the first message is accepted into a message collection. The collection expiry overrides any individual event handler timers. When the collection expiry timeout has passed for a message collection, the incomplete message collection is propagated to the **Expire** terminal. Connect appropriate processing nodes to the **Expire** terminal, to handle any expired message collections in your message flow.

To configure a collection expiry:

1. Switch to the Broker Application Development perspective.
2. Open the message flow with the Collector node.
3. Right-click the Collector node and select **Properties**.
4. Click on the **Basic** tab.
5. In Collection Expiry, enter a time in seconds for the collection expiry timeout.

Next: Configure the collection name.

Setting the collection name

You can set a default name, or use a correlation string, for the name of your message collections, by using the Collection name property on the Collector node.

Before you start:

To carry out this task, you must first have completed the following tasks:
- "Creating a message flow project" on page 256
- "Creating a flow that uses message collections" on page 685

Each message collection that is produced by the Collector node has a name. The collection name is the value that is associated with the CollectionName attribute in the message collection tree structure. Each message collection has only one name.

You can either use Collection name to set a default name to be used for each message collection, or you can use the event handler properties to create a correlation string to use for the message collection name. You can use the correlation string to generate a unique name for the message collection, based on the content of the input messages. To use the correlation string for the collection name, you must enter the wildcard symbol *. If you leave Collection name blank, or if it is set to * and the value of the correlation string is empty, the CollectionName attribute of the message collection is set to an empty value.

Any * characters in the collection name are replaced with the correlation string. The correlation string for each message collection is also copied into the local

environment message that is associated with the propagated message collection. The location of the correlation string in the local environment is `Wildcard/WildcardMatch`.

To configure the message collection name:

1. Switch to the Broker Application Development perspective.
2. Open the message flow with the Collector node.
3. Right-click the Collector node and select **Properties**.
4. Click the **Basic** tab.
5. For the Collection name property, enter a name for the message collections that are generated by the Collector node. If you have set a value for Correlation path on your input terminals, you can use the * for the Collection name property to substitute the correlation string into the collection name value. Leave the collection name blank if you want to set your message collection name to an empty value.

Next: "Setting the event coordination property."

Setting the event coordination property

Use the *Event coordination* property for controlling how message collections are propagated from the Collector node.

Before you start:

To complete this task, you must have completed the following tasks:

- "Creating a message flow project" on page 256
- "Creating a flow that uses message collections" on page 685

In addition to the dynamic input terminals that you can add to the Collector node, there is a static input terminal called **Control**. The purpose of this terminal is to allow an external resource to trigger the output from the collector node. Details are controlled through the Event coordination property settings.

Incomplete message collections that have exceeded the value for the *Collection expiry* timeout are immediately propagated to the Expire terminal, regardless of how you configure the *Event coordination* property.

To configure *Event coordination*:

1. Switch to the Broker Application Development perspective.
2. Open the message flow with the Collector node.
3. Right-click the Collector node and select **Properties**.
4. Click on the **Advanced** tab.
5. Set the *Event coordination* property on the Collector node. Select from the following options:
   - If you select `Disabled`, messages to the **Control** terminal are ignored and message collections are propagated when they are complete.
   - If you select `All complete collections`, complete message collections are held on a queue. When a message is received on the **Control** terminal, all message collections on the queue are propagated to the **Out** terminal.
   - If you select `First complete collection`, complete message collections are held on a queue. When a message is received on the **Control** terminal, the first message collection on the queue is propagated to the **Out** terminal. If

the queue is empty when a message arrives on the **Control** terminal, the next message collection that is completed is propagated to the **Out** terminal.

You have completed configuration of the Collector node.

Next: if you have configured your Collector node to use control messages, see "Using control messages with the Collector node" on page 694.

Setting the persistence mode property

Use the Persistence Mode property to control whether incomplete message collections are stored persistently on the Collector node's queues.

Before you start:

To complete this task, you must have completed the following tasks:
- "Creating a message flow project" on page 256
- "Creating a flow that uses message collections" on page 685

The storage of incoming messages and the Collector node state is handled internally by using WebSphere MQ queues. By default, incomplete message collections are stored non-persistently, which means that incomplete message collections persist if you restart your broker, but not if you restart your queue manager.

You can use the Persistence Mode property on the Collector node to store incomplete message collections on a queue persistently. If you set the Persistence Mode property to Persistent, incomplete message collections are not lost if you restart your queue manager. However, if you do set the property to Persistent, the overall performance of the Collector node might be affected.

To configure the Persistence Mode:

1. Switch to the Broker Application Development perspective.
2. Open the message flow that contains the Collector node.
3. Right-click the Collector node and select **Properties**.
4. Click the **Advanced** tab.
5. Set the Persistence Mode property on the Collector node to one of the following values.
   - If you select Non Persistent, messages and collection state are stored by the broker queue manager as non-persistent messages.
   - If you select Persistent, messages and collection state are stored by the broker queue manager as persistent messages.

You have completed configuration of the Collector node.

Setting the configurable service property

Use the *Configurable service* property to specify a Collector configurable service that sets the values of properties at run time.

Before you start:

To complete this task, you must have completed the following tasks:
- "Creating a message flow project" on page 256
- "Creating a flow that uses message collections" on page 685

You can use the *Configurable service* property on the Collector node to specify the name of a Collector configurable service to be used by the node. The Collector configurable service has the following properties:

queuePrefix
The queue prefix used to specify the queues in which events and collections are stored.

collectionExpirySeconds
The expiry time (in seconds) of a collection. This property overrides the value specified by the Collection expiry property on the Collector node.

To configure the *Configurable service* property:

1. Switch to the Broker Application Development perspective.
2. Open the message flow with the Collector node.
3. Right-click the Collector node and select **Properties**.
4. Click on the **Advanced** tab.
5. Specify the name of the Collector configurable service in the *Configurable service* field.

Using control messages with the Collector node

You can send control messages to the Collector node in order to control how complete message collections are propagated to other nodes in your message flow.

Before you start:

To complete this task, you must have completed the following tasks:
- "Creating a message flow project" on page 256
- "Creating a flow that uses message collections" on page 685
- "Setting the event coordination property" on page 692

You can control when complete message collections are propagated to other nodes for processing, using messages sent to the **Control** terminal. The exact behavior depends on the settings that you have chosen for the *Event coordination* property on the Collector node. If you want to use control messages to propagate completed messages collections you must set the *Event coordination* property to one of the following:

- `All complete collections`
- `First complete collection`

In these cases, the complete message collections are held on a queue until a control message is received. If you set *Event coordination* to `All complete collections`, all the message collections held on the queue are propagated to the **Out** terminal. If you set *Event coordination* to `First complete collection`, only the first message collection on the queue is propagated to the **Out** terminal. If there are no complete message collections on the queue, the next message collection to complete is immediately propagated to the **Out** terminal.

Incomplete message collections that have exceeded the value for *Collection expiry* are immediately propagated to the **Expire** terminal regardless of the setting of *Event coordination*.

If you want to propagate any complete message collections after a set amount of time for further processing, connect a TimeoutNotification node to the **Control** terminal of the Collector node. You can use the TimeoutNotification node to send a

control message to propagate the message collections to ensure that messages are processing within a reasonable time, or to schedule processing tasks.

For more information about driving a message flow using the TimeoutNotification node, see "Automatically generating messages to drive a flow" on page 702.

Alternatively, you can propagate complete message collections using a message from another application or message flow by connecting an input node to the **Control** terminal of the Collector node.

You can send any message to the **Control** terminal of the Collector node. The message received on the **Control** terminal is not examined by the broker and is discarded on receipt.

Configuring the storage of events for Collector nodes

You can use a Collector configurable service to control the storage of events for Collector nodes.

By default, the storage queues used by all Collector nodes are:

- SYSTEM.BROKER.EDA.EVENTS
- SYSTEM.BROKER.EDA.COLLECTIONS

These queues are also used by the Resequence node.

However, you can control the queues that are used by different Collector nodes by creating alternative queues containing a *QueuePrefix*, and using a Collector configurable service to specify the names of those queues for storing events.

Follow these steps to specify the queues that are used to store event state, and to set the expiry for the collection:

1. Create the storage queues that will be used by the Collector node. The following queues are required:
   - SYSTEM.BROKER.EDA.*QueuePrefix*.EVENTS
   - SYSTEM.BROKER.EDA.*QueuePrefix*.COLLECTIONS

   The *QueuePrefix* can contain any characters that are valid in a WebSphere MQ queue name, but must be no longer than 8 characters and must not begin or end with a period (.). For example, `SET1` and `SET.1` are valid queue prefixes, but `.SET1` and `SET1.` are invalid.

   If you do not create the storage queues, WebSphere Message Broker creates the set of queues (based on the default queues) when the node is deployed. If the queues cannot be created, the message flow is not deployed.
2. Use the mqsicreateconfigurableservice command to create a Collector configurable service. You can create a configurable service to be used with either a specific collection or with all collections in an execution group.
   a. If you are creating a configurable service to be used with a specific collection, ensure that the name of the configurable service is the same as the name that you specify in the Configurable service property on the Collector node (see step 3). If you are creating a configurable service to be used with all collections in the execution group, ensure that the configurable service has the same name as the execution group.
   b. Set the **Queue prefix** property to the required value.
   c. Optionally, set the **Collection expiry** property.

For example, create a Collector configurable service called myCollectorService, which uses queues prefixed with SYSTEM.BROKER.EDA.SET1, and with a collection expiry of 60 seconds:

```
mqsicreateconfigurableservice WBRK6_DEFAULT_BROKER -c Collector -o myCollectorService
-n queuePrefix,collectionExpirySeconds -v SET1,60
```

You can use the mqsideleteconfigurableservice command to delete the Collector configurable service. However, the storage queues are not deleted automatically when the configurable service is deleted, so you must delete them separately.

For more information, see Configurable services properties

3. In the Collector node:
   a. If the configurable service is to be used for a specific collection, specify the name of the configurable service in the Configurable service property on the **Advanced** tab; for example, myCollectorService. If you do not set the Configurable service property, and if a configurable service exists with the same name as the execution group, that configurable service is used.
   b. Optional: Use the mqsichangeproperties and mqsireportproperties commands to change or view the properties of the configurable service.

The properties for the configurable service are not used by the broker until you restart or redeploy the message flow, or restart the broker.

Configuring timeout flows

Use the TimeoutControl and TimeoutNotification nodes in message flows to process timeout requests or to generate timeout notifications at specified intervals.

The following topics show how these nodes can be used in a message flow:

- "Sending timeout request messages"
- "Sending a message after a timed interval" on page 698
- "Sending a message multiple times after a specified start time" on page 700
- "Automatically generating messages to drive a flow" on page 702
- "Configuring the storage of events for timeout nodes" on page 655

Sending timeout request messages

To set a controlled timeout, send a message with a set of elements with well known names to a TimeoutControl node. These elements control the properties of the timeout to be created or deleted.

Elements and format

The following example shows the elements and format of a timeout request message, showing the well known names and permissible values.

```
<TimeoutRequest>
  <Action>SET | CANCEL</Action>
  <Identifier>String (any alphanumeric string)</Identifier>
  <StartDate>String (TODAY | yyyy-mm-dd)</StartDate>
  <StartTime>String (NOW | hh:mm:ss)</StartTime>
  <Interval>Integer (seconds)</Interval>
  <Count>Integer (greater than 0 or -1)</Count>
  <IgnoreMissed>TRUE | FALSE</IgnoreMissed>
  <AllowOverwrite>TRUE | FALSE</AllowOverwrite>
</TimeoutRequest>
```

Message fields

The following table describes the fields in the message. The column headed M indicates whether the property is mandatory, and the column headed C indicates whether the property is configurable.

Property	M	C	Default	Description
Action	Yes	No	None	Set this element to either SET or CANCEL. An error is generated if you omit this element or set it to a different value. If you set it to CANCEL, the only other element that is required is the Identifier, which must match the Identifier of the TimeoutRequest that is to be canceled.
Identifier	Yes	No	None	Enter an alphanumeric string. An error is generated if you omit this element.
StartDate	No	No	TODAY	Set this element to TODAY or to a date specified in yyyy-mm-dd format. The default value is TODAY.
StartTime	No	No	NOW	Set this element to NOW or to a time in the future specified in hh:mm:ss format. The default value is NOW. StartTime is assumed to be the broker's local time. The start time can be calculated by adding an interval to the current time. If a delay occurs between the node that calculates the start time and the TimeoutControl node, the start time in the message will have passed by the time it reaches the TimeoutControl node. If the start time is more than approximately five minutes in the past, a warning is issued and the TimeoutControl node rejects the timeout request. If the start time is less than five minutes in the past, the node processes the request as if it were immediate. Therefore, ensure that the start time in the timeout request message is now or a time in the future.
Interval	No	Yes	0	Set this element to an integer that specifies the number of seconds between propagations of the message. The default value is 0.
Count	No	Yes	1	Set this element to an integer value that is either greater than 0 or is -1 (which specifies a timeout request that never expires). The default value is 1.
IgnoreMissed	Yes	No	TRUE	Set this element to TRUE or FALSE to control whether timeouts that occur while either the broker or the timeout notification flow is stopped, are processed the next time that the broker or timeout notification flow is started. The default value is TRUE, which means that missed timeouts are ignored by the TimeoutNotification node when the broker or message flow is started. If this value is set to FALSE, the missed timeouts are all processed immediately by the TimeoutNotification node when the flow is started. You must set the Request Persistence property of the TimeoutControl node to Yes or Automatic (with the originating request message being persistent) for the stored timeouts to persist beyond the restart of the broker or the timeout notification flow.
AllowOverwrite	N	N	TRUE	Set this element to TRUE or FALSE, to specify whether subsequent timeout requests with a matching Identifier can overwrite this timeout request. The default value is TRUE.

How the TimeoutControl node uses these values

Set the Request Location on the TimeoutControl node to InputRoot.XML.TimeoutRequest to read these properties. If you want to obtain properties from a different part of your message, specify the appropriate

correlation name for the parent element for the properties. The correlation name for the parent element can be in the local environment.

For details of how the TimeoutControl uses these values, see "TimeoutControl node" on page 1281

Working with the predefined schema definition of an XML timeout request message

A predefined schema definition of an XML timeout request message is provided in the workbench.

Take the following steps to review the definition or to define it in a message set.

1. Create or select a message set project that contains the message set.
2. Create a new message definition file (**File** → **New** → **Message Definition File From...**).
3. Select **IBM supplied message** and click **Next**.
4. Expand the tree for **Message Brokers IBM supplied Message Definitions**.
5. Select **Message Broker Timeout Request** and click **Finish**.

Example XML timeout request message

The format used here is XML, but you can use any format that is supported by an installed parser.

```
<TimeoutRequest>
  <Action>SET</Action>
  <Identifier>TenTimes</Identifier>
  <StartDate>TODAY</StartDate>
  <StartTime>NOW</StartTime>
  <Interval>10</Interval>
  <Count>10</Count>
  <IgnoreMissed>TRUE</IgnoreMissed>
  <AllowOverwrite>TRUE</AllowOverwrite>
</TimeoutRequest>
```

For another example of a timeout request message, and for details of how to use the timeout nodes to add timeouts to message flows, see the timeout processing sample:

- Timeout Processing

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

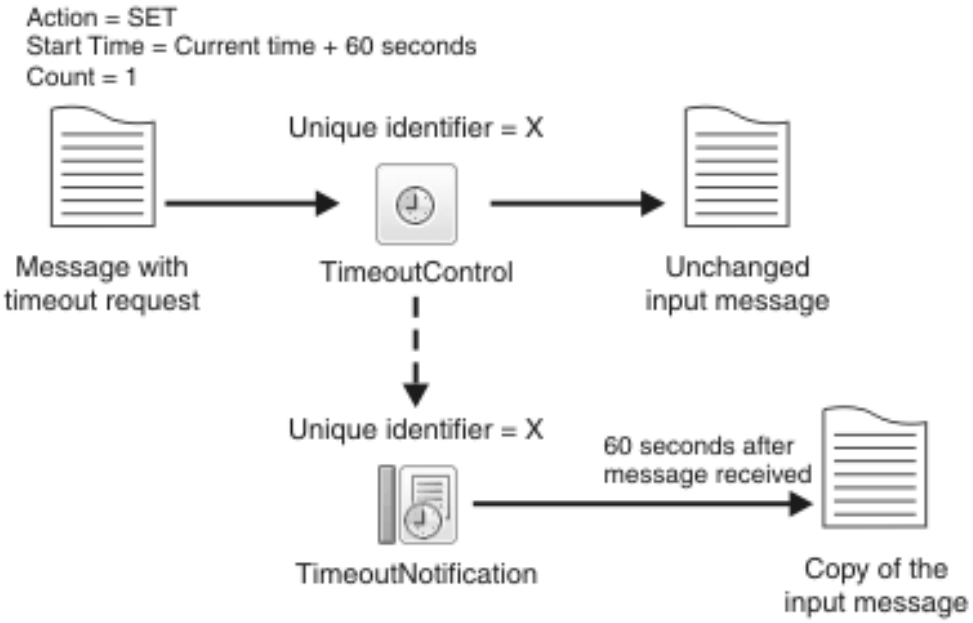
Sending a message after a timed interval

Use TimeoutControl and TimeoutNotification nodes to send a message into a message flow after a timed interval.

Aim

Use TimeoutControl and TimeoutNotification nodes to send a message into a message flow 60 seconds after the message is received.

Description of the flow



The diagram shows the path of a message that contains a timeout request through a TimeoutControl node. A TimeoutNotification node with an identifier matching the TimeoutControl node then processes the timeout request. The diagram also shows the message that the TimeoutNotification node produces after processing the timeout request.

The message comes into the TimeoutControl node with the following values set in the timeout request section of the message:

Action set to *SET*
Start Time set to *current time + 60*
Count set to *1*

The TimeoutControl node validates the timeout request; default values are assumed for properties that are not explicitly defined. The original message is then sent on to the next node in the message flow. If the request is valid, the TimeoutNotification node with the same Unique identifier as the TimeoutControl node propagates a copy of the message to the message flow 60 seconds after the message was received.

Timeout request messages are stored for processing on a queue used by the TimeoutNotification node. By default this queue is the SYSTEM.BROKER.TIMEOUT.QUEUE. However, you can use a Timer configurable service to specify an alternative timeout queue, which provides greater control over the storage of messages. For information about using an alternative timeout queue, see “Configuring the storage of events for timeout nodes” on page 655.

If a delay occurs between the node that calculates the start time and the TimeoutControl node, the start time in the message will have passed by the time it reaches the TimeoutControl node. If the start time is more than approximately five minutes in the past, a warning is issued and the TimeoutControl node rejects the timeout request. If the start time is less than five minutes in the past, the node

processes the request as if it were immediate. Therefore, ensure that the start time in the timeout request message is a time in the future.

Look at the following sample for further details on constructing this type of message flow.

- Timeout Processing

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

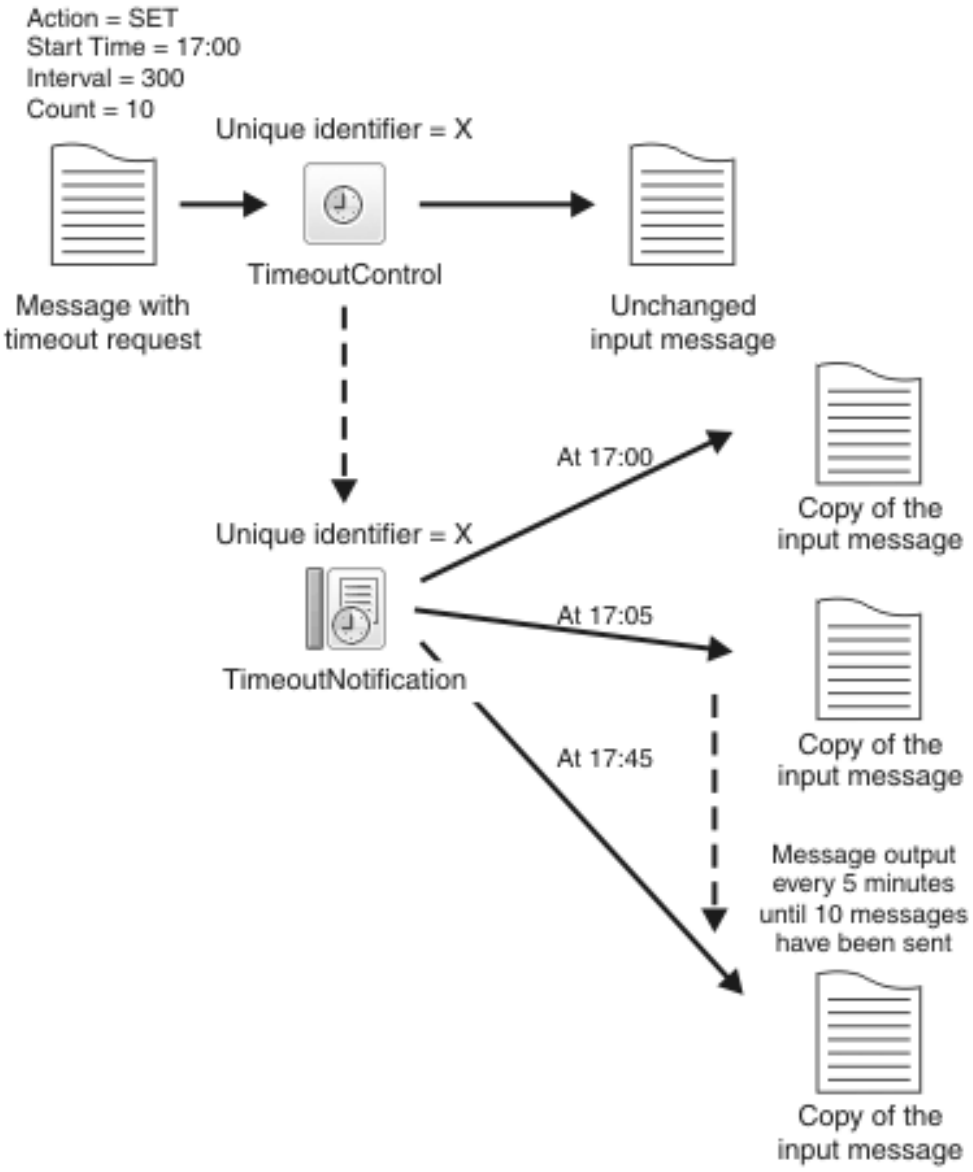
Sending a message multiple times after a specified start time

Use TimeoutControl and TimeoutNotification nodes to send a message into a message flow multiple times after a specified start time.

Aim

Use TimeoutControl and TimeoutNotification nodes to send a message into a message flow at 17:00 hours, then send the message again every 5 minutes until the message has been sent 10 times.

Description of the flow



The diagram shows the path of a message that contains a timeout request through a TimeoutControl node. A TimeoutNotification node with an identifier matching the TimeoutControl node then processes the timeout request. The diagram also shows the message that he TimeoutNotification node produces after processing the timeout request.

The message comes into the TimeoutControl node with the following values set in the timeout request section of the message:

Action set to *SET*
Start Time set to *17:00*
Interval set to *300*
Count set to *10*

The TimeoutControl node validates the timeout request; default values are assumed for properties that are not explicitly defined. The original message is then

sent on to the next node in the message flow. If the request is valid, the TimeoutNotification node with the same Unique identifier as the TimeoutControl node propagates a copy of the message to the message flow at 17:00. The message is sent again after an interval of 300 seconds, at 17:05. and every 300 seconds until the message has been sent 10 times, as the *Count* value in the timeout request specifies.

Look at the following sample for further details on constructing this type of message flow.

- Timeout Processing

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

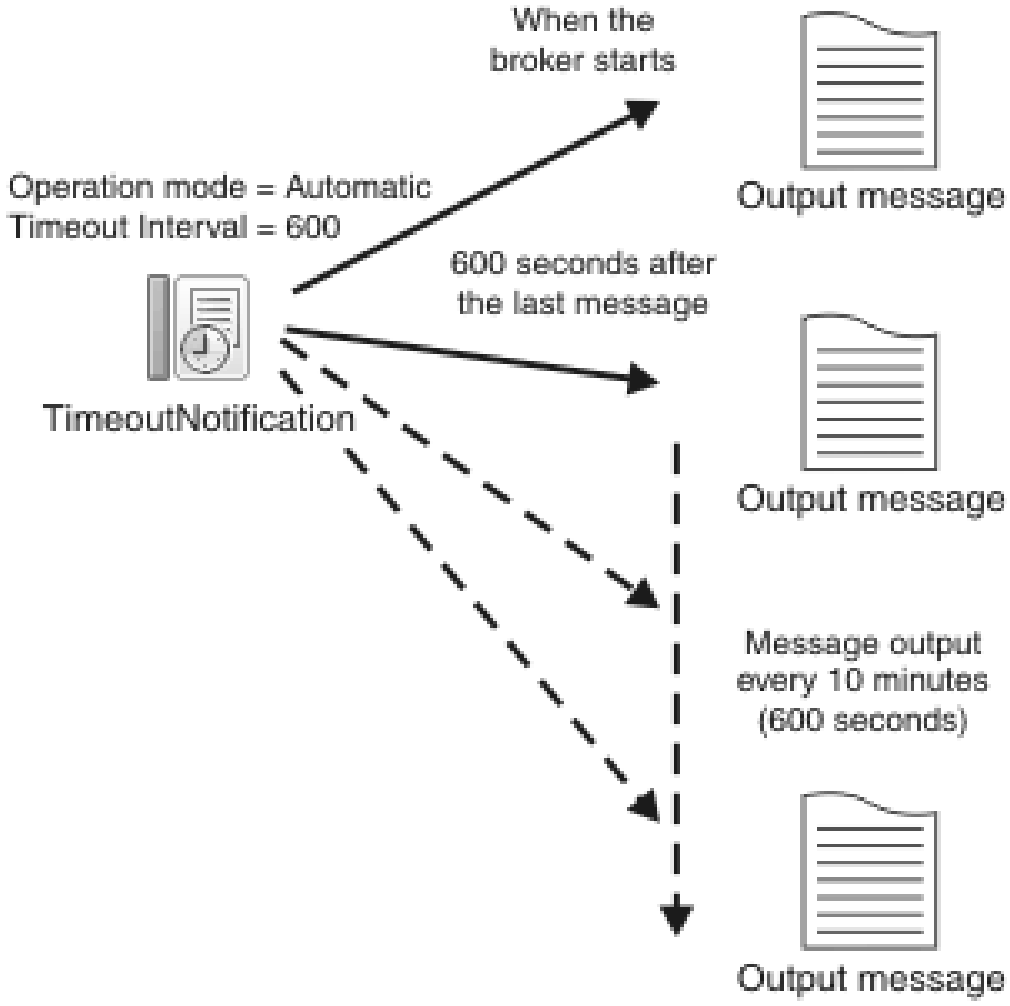
Automatically generating messages to drive a flow

Using the TimeoutNotification node to automatically send a message into a message flow.

Aim

Use the TimeoutNotification node to automatically send a message into a message flow every 10 minutes.

Description of the flow



The diagram shows a TimeoutNotification node automatically generating messages and propagating them every 10 minutes. To get the TimeoutNotification node to automatically generate messages, set the Operation Mode property of the node to automatic and specify a value for the Timeout Interval property. In this example the TimeoutNotification node has the following properties:

Operation Mode set to automatic
Timeout Interval set to 600

You can also use a Timer configurable service to control the timeout interval. A value set by the **Timeout interval** property of the Timer configurable service overrides the value specified on the TimeoutNotification node. For more information about the Timer configurable service, see Configurable services properties.

When the broker has started, the TimeoutNotification node sends a message every 10 minutes (600 seconds). This message contains only a properties folder and a LocalEnvironment folder. A Compute node can then process this message to create a more meaningful message.

In the above example, the relevant property is IgnoreMissed, and for an automatic timeout this is implicitly set to TRUE. This means that if one of the period events is missed the event will not be resent, but instead the broker will trigger the event on the next scheduled timeout. If you want to be notified when events are missed, set a controlled timeout instead. For details of the properties you can set for a controlled timeout, see "Sending timeout request messages" on page 696.

Look at the following sample for further details on constructing this type of message flow.

- Timeout Processing

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

Configuring the storage of events for timeout nodes

You can use a Timer configurable service to control the storage of events for TimeoutNotification and TimeoutControl nodes.

By default, the storage queue used by all timeout nodes is the SYSTEM.BROKER.TIMEOUT.QUEUE

However, you can control the queues that are used by different timeout nodes by creating alternative queues containing a *QueuePrefix*, and using a Timer configurable service to specify the names of those queues for storing events.

Follow these steps to specify the queue that is used to store event state:

1. Create the storage queue that will be used by the timeout nodes. The following queue is required:
   - SYSTEM.BROKER.TIMEOUT.*QueuePrefix*.QUEUE

   The *QueuePrefix* can contain any characters that are valid in a WebSphere MQ queue name, but must be no longer than 8 characters and must not begin or end with a period (.). For example, `SET1` and `SET.1` are valid queue prefixes, but `.SET1` and `SET1.` are invalid.

   If you do not create the storage queue, WebSphere Message Broker creates the queue (based on the default queue) when the node is deployed. If the queue cannot be created, the message flow is not deployed.
2. Use the mqsicreateconfigurableservice command to create a Timer configurable service. You can create a configurable service to be used with either specific timeout requests or with all timeout requests in an execution group.
   a. If the configurable service is to be used with specific timeout requests, create the configurable service with the same name as the Unique identifier property on the TimeoutNotification and TimeoutControl nodes. If the configurable service is to be used with all timeout requests in an execution group, create the configurable service with the same name as the execution group.
   b. Set the **Queue prefix** property to the required value.

   For example, create a Timer configurable service that uses a queue prefixed with SYSTEM.BROKER.TIMEOUT.SET1:

   ```
   mqsicreateconfigurableservice WBRK6_DEFAULT_BROKER -c Timer -o myTimer
   -n queuePrefix -v SET1
   ```

   You can use the mqsideleteconfigurableservice command to delete the Timer configurable service. However, the storage queue is not deleted automatically

when the configurable service is deleted, so you must delete it separately.For more information, see Configurable services properties.

3. In the TimeoutNotification and TimeoutControl nodes:
   a. Ensure that the name of the Timer configurable service is the same as the name specified in the Unique Identifier property on the **Basic** tab; for example, `myTimer`. If no Timer configurable service exists with the same name as the Unique Identifier, and if a configurable service exists with the same name as the execution group, that configurable service is used instead.
   b. Optional: Use the mqsichangeproperties and mqsireportproperties commands to change or view the properties of the configurable service.

The properties for the configurable service are not used by the broker until you restart or redeploy the message flow, or restart the broker.

Considering performance for timeout flows

When you design timeout flows, the decisions that you make can affect the performance of your brokers and applications.

You can use the timeout nodes TimeoutControl and TimeoutNotification in your message flows to control the way in which your message flows operate:

- Set the Operation Mode property of the TimeoutNotification node to Automatic. This setting causes a flow to be invoked at the interval that you specify in the Timeout Value property. If the downstream processing is intensive, and the flow is still busy when the next timeout occurs, the flow is not started for that timeout instance. The flow is notified to start again only if it is free when a particular timeout occurs.

  The value of the Additional Instances property of the message flow is ignored downstream of a TimeoutNotification node, and you cannot use this property to change the behavior of the flow.
- Use two associated flows to perform user-defined timeout processing. Set a timeout with a TimeoutControl node, and notify the flow using a TimeoutNotification node (which behaves like an input node to start a new message flow thread). If the downstream processing from the TimeoutNotification node is significant, requests that are set up in the TimeoutControl node can build up. You can specify that the timeout messages are generated only when the flow that starts with the TimeoutNotification node becomes free again.

  You cannot increase the Additional Instances property of the message flow if it starts with a TimeoutNotification node, therefore you cannot apply more threads to increase the capacity of the flow.

Although you can use a TimeoutNotification node to cause nodes in a message flow to poll for the next item of work, this approach forces a delay between each transaction, and typically does not provide an efficient solution. If you want to periodically check a resource for the next piece of work, and process it immediately, consider one or more of the following alternative solutions:

- Use a built-in input node.
- Write your own input node by using the user-defined node API (in Java or C).
- Consider purchasing an IBM or vendor-provided adapter which polls the subsystem you want, and triggers the flow.

A message flow that uses these options can process more work overall than it can if you implement a timeout solution, and incurs lower CPU cost, although your initial development costs might be slightly higher.

Configuring flows to handle WebSphere MQ message groups

WebSphere MQ allows multiple messages to be treated as a group, or as segments of one larger message. WebSphere Message Broker provides support for WebSphere MQ message grouping and partial support for message segmenting.

You can use the MQInput and MQOutput nodes to receive and send messages that are part of a WebSphere MQ message group. You can use the MQOutput node to send messages that are segments of a larger message.

For guidance about configuring the MQInput and MQOutput nodes to receive and send messages that are part of a WebSphere MQ message group, see:

- "Receiving messages in a WebSphere MQ message group"
- "Sending messages in a WebSphere MQ message group" on page 708
- "Sending message segments in a WebSphere MQ message" on page 709

For more information about WebSphere MQ message groups, see the *Application Programming Guide* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.

Receiving messages in a WebSphere MQ message group

You can configure the MQInput node to receive messages that are in a WebSphere MQ message group.

The following properties on the MQInput node control the processing of messages in a WebSphere MQ message group:

- Logical order
- Order mode
- All messages available
- Commit by message group

To ensure that your message flow receives group messages in the order that has been assigned by the sending application, select Logical order. If you do not select this option, messages that are sent as part of a group are not received in a predetermined order. This property maps to the MQGMO_LOGICAL_ORDER option of the MQGMO of the MQI. More information about the options to which this property maps is available in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.

If you specify a value of By Queue Order on the Order mode property, the message flow processes the messages in the group in the order that is defined by the queue attributes; this order is guaranteed to be preserved when the messages are processed. This behavior is identical to the behavior that is exhibited if the Additional instances property is set to zero. The message flow processes the messages on a single thread of execution, and a message is processed to completion before the next message is retrieved from the queue. If you do not specify this value, it is possible that multiple threads within a single message flow are processing multiple messages, and the final message in a group, which

prompts the commit or roll back action, is not guaranteed to be processed to completion after all other messages in the group.

To ensure that only a single instance of the message flow processes the group messages in the order that has been assigned by the sending application, select Logical order **and** specify a value of By Queue Order on the Order mode property.

If you select All messages available, message retrieval and processing is performed only when all messages in a single group are available. This means that messages in a group are not received until all the messages in the group are present on the input queue. It is good practice to select this check box when your message flow needs to process group messages. If you do not select this check box, the message flow receives the messages as they arrive on the input queue; if a message in the group fails to arrive on the input queue, the message flow waits for it and cannot process any further messages until this message arrives. This property maps to the MQGMO_ALL_MESSAGES_AVAILABLE option of the MQGMO of the MQI. More information about the options to which this property maps is available in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.

If you select Commit by message group, message processing is committed only after the final message of a group has been received and processed. If you leave this check box cleared, a commit is performed after each message has been propagated completely through the message flow. This property is relevant only if you have selected Logical order. It is good practice to select this check box together with the All messages available check box because this ensures that the complete message group is retrieved and processed in the same unit of work without risk of the message flow waiting indefinitely for messages in the group to arrive on the input queue.

To ensure that the message flow that processes group messages does not wait for unavailable messages:

- Avoid having multiple message flows reading from the same input queue when group messages are being retrieved.
- Avoid deploying additional instances of a flow that retrieves group messages.
- Avoid using expired messages in message groups.
- When expired messages are to be used, ensure either that all messages have the same expiry time or that the first message in the group is set to expire before the rest of the group. If the first message in a group cannot be retrieved, the group can never be started in logical order.

If a message flow waits for a group message that does not arrive within the wait interval, a BIP2675 warning message is issued. From that point on, the message flow always attempts to retrieve the next group message and does not process any other input messages until the next group message is received.

Therefore, if the expected group message does not arrive, or has expired, the message flow must be stopped manually, and any incomplete message group cleared from the input queue.

A message flow cannot receive all the messages in a group in one operation.

If you specify a value of Yes or No on the Transaction mode property, all the segments in a message are received in the message flow as a single message. As a result, the message flow might receive very large messages which might lead to

storage problems in the broker. If you specify a value of Automatic on the Transaction mode property, message segments are received as individual messages.

Sending messages in a WebSphere MQ message group

The MQOutput node can send multiple messages that form a WebSphere MQ message group. Configure a Compute or JavaCompute node to set the MQMD fields to specify message group options.

The message flow must set the following MQMD fields:

- **GroupId**
- **MsgSeqNumber**
- **MsgFlags**

You can use the following example of ESQL code in a Compute node, which shows how to set these fields:

```
DECLARE MSGNUM INT 0;
  DECLARE MSGTOTAL INT 5;
  WHILE MSGNUM < MSGTOTAL DO
    SET MSGNUM = MSGNUM + 1;
    CALL CopyMessageHeaders();
    -- Manually set the groupId since we cant ask the queue manager to generate one.
    -- the UUIDASBLOB function could be used here to generate one, but this must be done
    -- outside the loop to keep the same groupId throughout!
    SET OutputRoot.MQMD.GroupId = X'000000000000000000000000000000000000000000000001';
    SET OutputRoot.MQMD.MsgSeqNumber = MSGNUM;
    SET OutputRoot.MQMD.MsgFlags = MQMF_MSG_IN_GROUP;
    IF (MSGNUM = MSGTOTAL) THEN
      SET OutputRoot.MQMD.MsgFlags = MQMF_LAST_MSG_IN_GROUP;
    END IF;
    SET OutputRoot.XML.TestCase = MSGNUM;
    PROPAGATE;
  END WHILE;
  RETURN FALSE;
```

You can use the following example of Java code in a JavaCompute node, which shows how to set these fields:

```
public class mq_group_java_JavaCompute extends MbJavaComputeNode {
      public void evaluate(MbMessageAssembly assembly) throws MbException {
        MbOutputTerminal out = getOutputTerminal("out");
        int MSGTOTAL = 5;
        for(int MSGNUM = 1 ; MSGNUM <= MSGTOTAL;MSGNUM++){
           MbMessage newMessage = new MbMessage();
           MbElement root = newMessage.getRootElement();
           MbElement mqmd = root.createElementAsFirstChild("MQMD");
           mqmd.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,
           "GroupId", new byte[]{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1});
           mqmd.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "MsgSeqNumber", MSGNUM);
           if (MSGNUM == MSGTOTAL){
               mqmd.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,
               "MsgFlags", 16); // MQMF_LAST_MSG_IN_GROUP 0x00000010
           }
           else{
               mqmd.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,
               "MsgFlags", 8);  // MQMF_MSG_IN_GROUP      0x00000008
           }
           MbElement xmlnsc = root.createElementAsLastChild("XMLNSC");
           MbElement body = xmlnsc.createElementAsLastChild(MbElement.TYPE_NAME,"TestCase",null);
           body.setValue(MSGNUM);
           MbMessageAssembly newAssembly = new MbMessageAssembly(assembly,newMessage);
```

```
            out.propagate(newAssembly);
        }
    }
}
```

If the message flow is sending multiple messages from one input message, it can create a **GroupId** value, increment the **MsgSeqNumber** value, and set the **MsgFlags** field. The examples of ESQL and Java code show how you can set these values. However, if the message flow is sending multiple messages from more than one input message, it must store the **GroupId** and **MsgSeqNumber** values between flow instances, which is achieved by using shared variables.

For more information about message grouping, see the *Application Programming Guide* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online. For more information about the WebSphere MQ fields that are significant in message grouping, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.

Sending message segments in a WebSphere MQ message

The MQOutput node can send multiple message segments that form a WebSphere MQ message. Configure a Compute node to set the MQMD fields to specify message segment options.

You can either select Segmentation allowed on the node, or set the required fields in the MQMD in the message flow:

- GroupId
- MsgFlags
- Offset

Use the example ESQL code in "Sending messages in a WebSphere MQ message group" on page 708 and change the code to set these fields.

For more information about message grouping and segmentation, see the *Application Programming Guide* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online. For more information about the WebSphere MQ fields that are significant in message grouping and segmentation, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.

HTTP proxy servlet overview

The HTTP proxy servlet is a Java servlet that you can use in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat, to receive HTTP requests from web services client applications, to replace the support provided by the broker HTTP listeners.

By using the HTTP proxy servlet in an external web servlet container, you can support a larger number of concurrent HTTP sessions, high availability, load distribution, and access to the broker from multiple IP addresses and ports.

The HTTP proxy servlet supports SSL (HTTPS) secure protocol when it is deployed in a properly configured web servlet container.

The proxy servlet is available as part of the runtime in IBM WebSphere Message Broker Version 6.1 Fix Pack 3 (6.1.0.3) and above, and is available in SupportPac IE01 for WebSphere Message Broker Version 5.0 and 6.0.

For a detailed description of the proxy servlet, its location in the runtime directory path, and its function, see: "HTTP proxy servlet; proxy servlet component" on page 716

Before you install and test the HTTP proxy servlet, ensure you understand the following concepts:

- "HTTP traffic handling in WebSphere Message Broker"
- "HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 711
- "HTTP proxy servlet; descriptions of required components" on page 715

When you have gained an understanding of the proxy servlet concept, read the following topics to help you install and test the HTTP proxy servlet:

- "Installing the proxy servlet" on page 720
- "Testing the proxy servlet" on page 730

HTTP traffic handling in WebSphere Message Broker

Before reading about HTTP traffic handling using the proxy servlet in an external Web container, ensure that you understand the basic structure of HTTP traffic handling in WebSphere Message Broker.

WebSphere Message Broker supports Web services requests over HTTP. Brokers have internal HTTP listeners that receive requests, which are propagated to message flows that contain the following nodes:

- HTTPInput
- HTTPReply
- SOAPInput
- SOAPReply

WebSphere Message Broker has two types of listeners:

- A listener that is started and managed by the broker, which you can use for messages for the HTTP nodes. This listener supports two ports, one for HTTP and one for HTTPS (HTTP over SSL) messages.
- A listener that is started and managed by an execution group (an embedded listener), which you can use for messages for the SOAP nodes. This listener also supports two ports for HTTP and HTTPS messages.

HTTP traffic handling is based on node type:

- HTTP nodes: Connections are made into the broker HTTP listener, which places requests on the SYSTEM.BROKER.WS.INPUT queue, from which the HTTPInput nodes read the data. After the flow has reached an HTTPReply node, the reply data is placed on the SYSTEM.BROKER.WS.REPLY queue, where the data is read by the HTTP listener and sent back to the HTTP client.

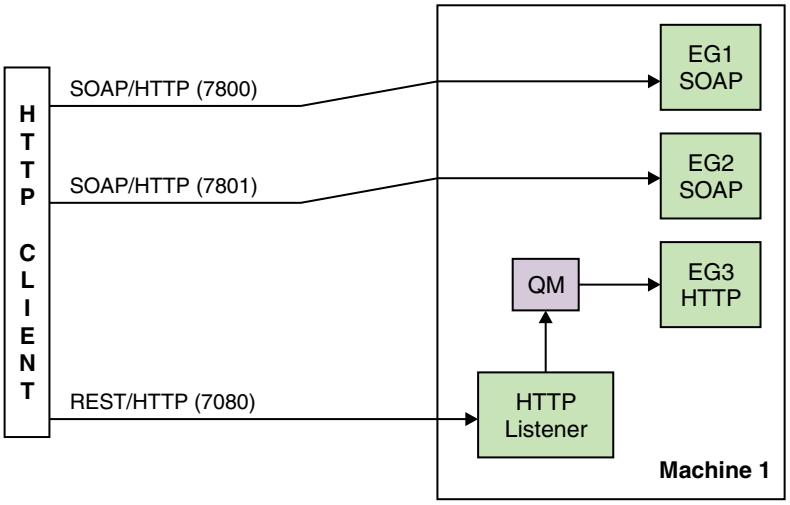
  If you want HTTP nodes to handle HTTPS messages, you must update the broker configuration to define a second port for HTTPS, and set the property to enable HTTPS processing, by using the mqsichangeproperties command. You can also change the port or ports on which the listener is listening by using this command.

Because WebSphere MQ queues are used to couple the HTTP listener to the message flows, requests received by the HTTP listener can be processed by any message flow in any execution group (provided that the Web address selector matches), and the reply can come back from any execution group.

- SOAP nodes: Connections are made directly to the execution group listener, and requests are passed to the SOAPInput nodes in that execution group. The HTTP data is passed to the SOAPInput node, processed in the flow, and sent back directly from the SOAPReply node.

  Because the network connection is made to a particular execution group, the reply must be sent back from that execution group.

The following diagram is a visual representation of the HTTP traffic handling connections based on node type:



Before you install and test the HTTP proxy servlet, ensure that you understand the following concepts:

- "HTTP proxy servlet overview" on page 709
- "HTTP traffic handling by using the proxy servlet in an external web servlet container"
- "HTTP proxy servlet; descriptions of required components" on page 715

When you have gained an understanding of the proxy servlet concept, read the following topics to help you to install and test the HTTP proxy servlet:

- "Installing the proxy servlet" on page 720
- "Testing the proxy servlet" on page 730

HTTP traffic handling by using the proxy servlet in an external web servlet container

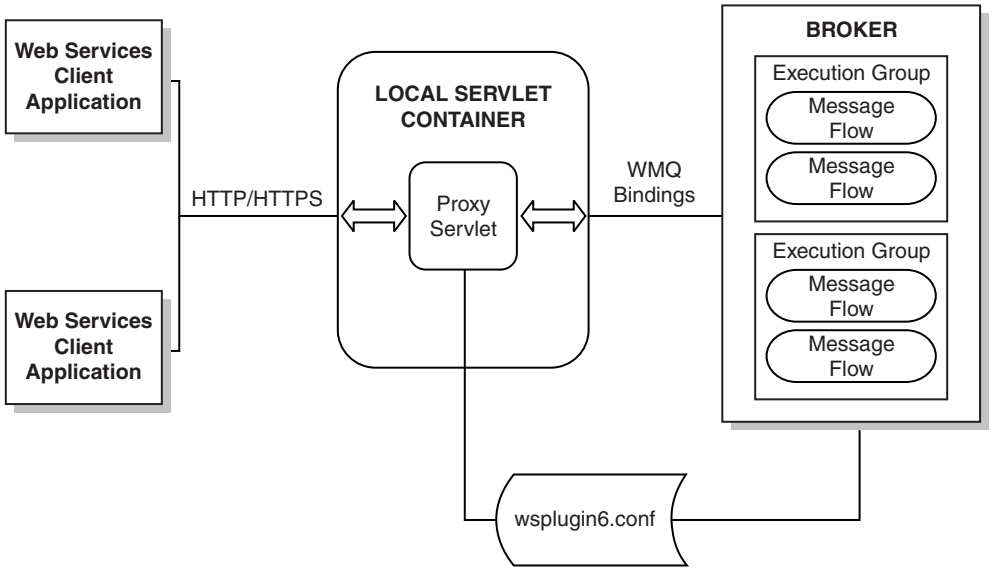
Before you install and test the HTTP proxy servlet, ensure that you understand the structure of HTTP traffic handling by using the proxy servlet in an external web servlet container, such as IBM WebSphere Application Server or Apache Tomcat.

The proxy servlet is a Java servlet that produces the function of the broker HTTP listeners in an external web servlet container, such as WebSphere Application Server or Apache Tomcat. After the proxy servlet is deployed and running on the web servlet container, it uses the HTTP listener of the container to receive HTTP requests. If the web servlet container is configured to support SSL (HTTPS), web

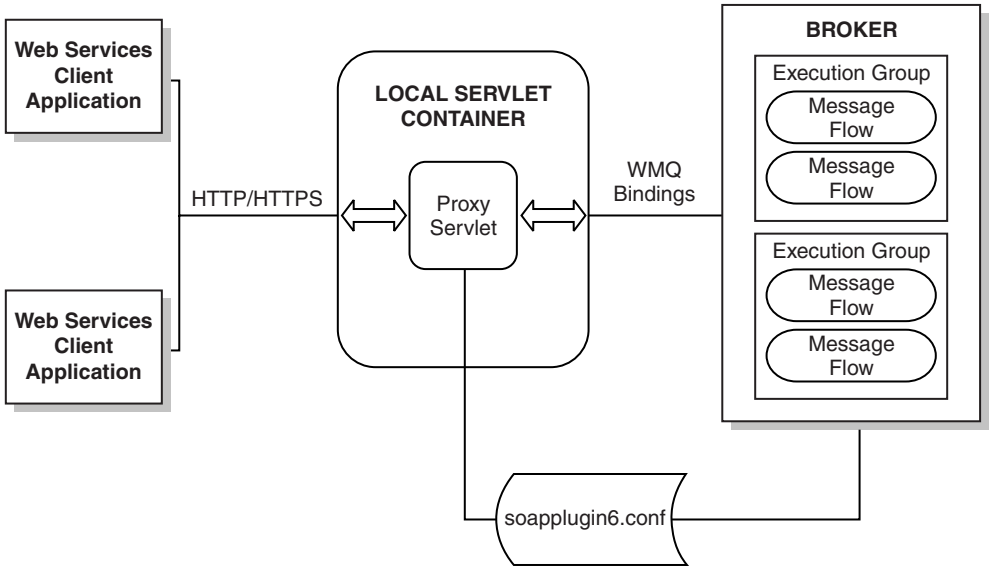
services requests are received by the message flows by using a secure communications protocol. For more information, see: “HTTP proxy servlet; proxy servlet component” on page 716

Components and configurations supported by the proxy servlet:

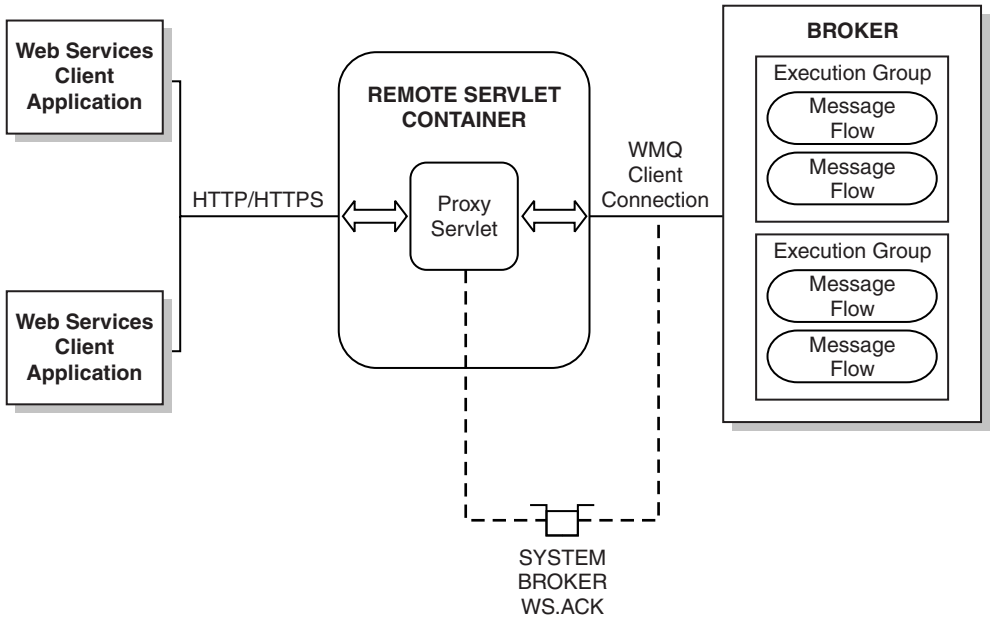
The following figures show the components and configurations that are supported by the proxy servlet, for descriptions of the components listed in these figures, see: “HTTP proxy servlet; descriptions of required components” on page 715



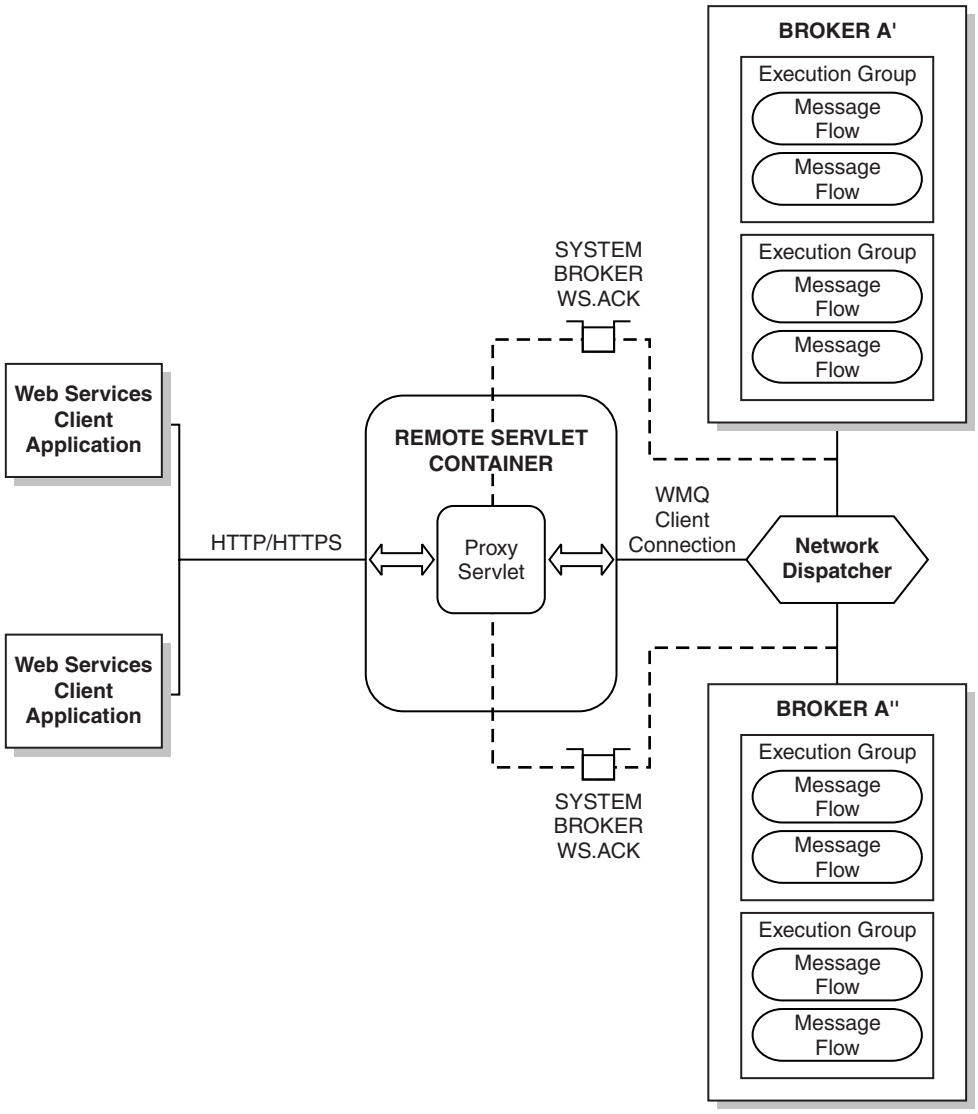
In the above figure the proxy servlet is running on the same server as the message broker. The proxy servlet connects to the broker queue manager by using bindings mode (local connection) and the servlet is configured to access only HTTP nodes and therefore has been configured to access the HTTP nodes configuration file `wsplugin6.conf`.



In the above figure the proxy servlet is running on the same server as the message broker. The proxy servlet connects to the broker queue manager by using bindings mode (local connection) and the servlet is configured to access only SOAP nodes and therefore has been configured to access the SOAP nodes configuration file `soapplugin6.conf`.



In the above figure the proxy servlet is running on a remote server to the message broker. The proxy servlet connects to the broker queue manager by using client mode (remote connection) and the servlet can be configured to access HTTP or SOAP nodes and the HTTP or SOAP node configuration is retrieved from the SYSTEM.BROKER.WS.ACK queue.



In the above figure the proxy servlet is configured to load balance WebSphere MQ connections across multiple message brokers. Network dispatchers or load balancers are required for this configuration to work.

When the proxy servlet is configured to connect to multiple brokers, the brokers must be identical clones of each other, which means that the same HTTP and SOAP flows are deployed with the same web addresses.

The proxy servlet sends the HTTP requests over the WebSphere MQ connections trying to distribute the load between the active broker connections.

Before you install and test the HTTP proxy servlet, ensure you understand the following concepts:

- "HTTP proxy servlet overview" on page 709
- "HTTP traffic handling in WebSphere Message Broker" on page 710

When you have gained an understanding of the proxy servlet concept, read the following topics to help you install and test the HTTP proxy servlet:

- "Installing the proxy servlet" on page 720
- "Testing the proxy servlet" on page 730

HTTP proxy servlet; descriptions of required components

The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, WebSphere Message Broker, and web services clients.

Ensure that you are familiar with the following components and concepts required by the proxy servlet:

- "HTTP proxy servlet; message flows component"
- "HTTP proxy servlet; proxy servlet component" on page 716
- "HTTP proxy servlet; servlet container component" on page 717
- "HTTP proxy servlet; web addresses component" on page 717
- "HTTP proxy servlet; WebSphere Message Broker component" on page 719
- "HTTP proxy servlet; web services clients component" on page 719

Before you install and test the HTTP proxy servlet, ensure you understand the following concepts:

- "HTTP proxy servlet overview" on page 709
- "HTTP traffic handling in WebSphere Message Broker" on page 710
- "HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 711

When you have gained an understanding of the proxy servlet concept, read the following topics to help you install and test the HTTP proxy servlet:

- "Installing the proxy servlet" on page 720
- "Testing the proxy servlet" on page 730

HTTP proxy servlet; message flows component

The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, WebSphere Message Broker, and web services clients. Ensure that you are familiar with the message flows component.

Message flows execute the functions of transforming, logging, enriching, and routing messages. Message flows can use HTTP or SOAP input nodes to receive requests and HTTP or SOAP reply nodes to send responses. Each HTTP or SOAP input node has a web address configured in the node properties. The HTTP or SOAP input nodes only receive request messages addressed to the configured web address.

The proxy servlet is not aware of the execution groups that the message flows are deploying.

Before you install and test the HTTP proxy servlet, ensure you understand the following concepts:

- "HTTP proxy servlet overview" on page 709
- "HTTP traffic handling in WebSphere Message Broker" on page 710
- "HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 711
- "HTTP proxy servlet; descriptions of required components"

When you have gained an understanding of the proxy servlet concept, read the following topics to help you install and test the HTTP proxy servlet:

- "Installing the proxy servlet" on page 720
- "Testing the proxy servlet" on page 730

HTTP proxy servlet; proxy servlet component

The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, WebSphere Message Broker, and web services clients. Ensure that you are familiar with the proxy servlet component.

A *Proxy servlet* is a Java Web Application Archive (WAR) file that is part of the runtime environment in WebSphere Message Broker Version 6.1 Fix Pack 3 (6.1.0.3) and above, and can be found in the following directory:

WMB61_runtime_install_path/tools. Where *WMB61_runtime_install_path* specifies the name of your runtime installation directory.

The proxy servlet is a Java servlet that receives HTTP requests. The proxy servlet matches the received web address with the web address that the HTTP or SOAP input nodes are monitoring, then passes the HTTPRequest to the correct HTTP or SOAP input node flow using WebSphere MQ.

The proxy servlet receives response messages from the HTTP or SOAP reply nodes and sends them back to the client applications over HTTP or HTTPS. The message broker has several internal WebSphere MQ queues, SYSTEM.BROKER.WS.* queues, that are used for the communication between the proxy servlet and the HTTP or SOAP input and reply nodes.

Each HTTP or SOAP input node monitors the arrival of requests associated to specific web addresses. The message broker has internal configuration files, and an internal WebSphere MQ queue SYSTEM.BROKER.WS.ACK, that contain the list of web addresses that are monitored by the different HTTP or SOAP input nodes in message flows deployed on any execution groups. The proxy servlet accesses the internal file, or queue, to match the web addresses received in HTTP or HTTPS requests with the web addresses that the HTTP or SOAP input nodes that are waiting for.

The configuration files, or queue, have a unique correlation ID associated with each web address. The HTTP or SOAP input node uses this correlation ID to get the messages from the internal queue SYSTEM.BROKER.WS.INPUT, and the proxy servlet uses the same correlation ID to put the messages on this queue. This is the mechanism used to correlate the incoming HTTP or HTTPS requests and the HTTP or SOAP input nodes in the message flows.

The HTTP or SOAP input node copies the WebSphere MQ input message ID in the LocalEnvironment.Destination.HTTP.RequestIdentifier to be used by the HTTP or SOAP reply node for WebSphere MQ output message correlation ID. The proxy servlet does a selective GET by correlation ID to the reply queue SYSTEM.BROKER.WS.REPLY, and receives the response messages.

The proxy servlet accepts GET and POST HTTP or HTTPS requests.

Before you install and test the HTTP proxy servlet, ensure you understand the following concepts:

- "HTTP proxy servlet overview" on page 709
- "HTTP traffic handling in WebSphere Message Broker" on page 710
- "HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 711
- "HTTP proxy servlet; descriptions of required components" on page 715

When you have gained an understanding of the proxy servlet concept, read the following topics to help you install and test the HTTP proxy servlet:

- "Installing the proxy servlet" on page 720
- "Testing the proxy servlet" on page 730

HTTP proxy servlet; servlet container component

The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, WebSphere Message Broker, and web services clients. Ensure that you are familiar with the servlet container component.

A *Servlet container* is the runtime environment for servlets and Java Server Pages (JSP). WebSphere Application Server and Apache Tomcat are two examples of servlet containers (or web containers) that are available. The proxy servlet can be deployed in a local servlet container that is running on the same server as message broker or on a remote servlet container that is running on a remote server to message broker. The servlet container must allow the proxy servlet to configure and call the WebSphere MQ classes for Java.

The servlet container provides the SSL (HTTPS) listener support for web services applications. SSL must be configured and available in the container. The proxy servlet does not have to be configured for SSL but it enforces HTTP requests over SSL if the HTTP or SOAP input node is configured to use SSL.

Before you install and test the HTTP proxy servlet, ensure you understand the following concepts:

- "HTTP proxy servlet overview" on page 709
- "HTTP traffic handling in WebSphere Message Broker" on page 710
- "HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 711
- "HTTP proxy servlet; descriptions of required components" on page 715

When you have gained an understanding of the proxy servlet concept, read the following topics to help you install and test the HTTP proxy servlet:

- "Installing the proxy servlet" on page 720
- "Testing the proxy servlet" on page 730

HTTP proxy servlet; web addresses component

The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, WebSphere Message Broker, and web services clients. Ensure that you are familiar with the web addresses component.

web addresses, or Universal Resource Locators (URLs), have an important role when HTTP or SSL (HTTPS) protocols are used. In WebSphere Message Broker, each HTTP or SOAP input node expects to receive requests from a specific web address, or web addresses when wildcard characters are used. The servlet container also

uses the web address to locate the servlets that are going to process the HTTP or HTTPS requests received by the listener in the container.

The proxy servlet passes the requests from the servlet container to the broker and vice versa. web addresses have the dual function of locating servlets and locating HTTP or SOAP input nodes. This affects the format of the web addresses used for the broker.

A web address is made up of the following structure:
`<schema>://<hostname>:<port>/<url_path>`

web address structure definition:
- *<schema>*
- *<hostname>*
- *<port>*
- *<url_path>*

<schema> is HTTP or HTTPS.

<hostname> is the hostname, or IP address, of the server where the servlet container is running.

<port> is the port number that the servlet container is listening on.

<url_path> is a series of tokens separated by slashes /. These are used to indicate the location of the servlet and the location of the HTTP or SOAP input nodes.

Because the *<url_path>* is used for mapping two resources (instead of one with the broker internal listener) the format of the web address will change when the proxy servlet is used.

The broker structure of the *<url_path>* is:
`/<url_path>=/<context_root>/<node_url_path>`

Broker *<url_path>* structure definition:
- *<context_root>*
- *<node_url_path>*
- *<port>*
- *<url_path>*

<context_root> is the *<url_path>* allocated to the proxy servlet by the container when the servlet is installed and deployed.

<node_url_path> is the part of the web address path that is added to make the web address unique to a specific HTTP or SOAP input node.

The entire *<url_path>* has to be configured in the properties of the HTTPInput node.

In some web servlet containers, it is possible to configure the proxy servlet to receive all the HTTP or HTTPS requests arriving to the container (*<context_root>* = "/*"). In this case, the existing web addresses in the HTTP nodes do not have to change when the proxy servlet is implemented.

Before you install and test the HTTP proxy servlet, ensure you understand the following concepts:

- "HTTP proxy servlet overview" on page 709
- "HTTP traffic handling in WebSphere Message Broker" on page 710
- "HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 711
- "HTTP proxy servlet; descriptions of required components" on page 715

When you have gained an understanding of the proxy servlet concept, read the following topics to help you install and test the HTTP proxy servlet:

- "Installing the proxy servlet" on page 720
- "Testing the proxy servlet" on page 730

HTTP proxy servlet; WebSphere Message Broker component

The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, Web addresses, WebSphere Message Broker, and Web services clients. Ensure that you are familiar with the WebSphere Message Broker component.

The WebSphere Message Broker is the runtime environment for the message flows. Message flows can receive requests over HTTP when there are HTTP or SOAP input nodes in the flow.

WebSphere Message Broker Version 6.1 has two types of listeners:

A type for HTTP nodes, and there is one instance of this listener type per broker, and another type for SOAP nodes, and there is one instance of this listener type per execution group.

Each listener has its unique TCP/IP ports to listen for HTTP and HTTPS requests.

The proxy servlet implements the HTTP or HTTPS listener function in an external servlet container.

Before you install and test the HTTP proxy servlet, ensure you understand the following concepts:

- "HTTP proxy servlet overview" on page 709
- "HTTP traffic handling in WebSphere Message Broker" on page 710
- "HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 711
- "HTTP proxy servlet; descriptions of required components" on page 715

When you have gained an understanding of the proxy servlet concept, read the following topics to help you install and test the HTTP proxy servlet:

- "Installing the proxy servlet" on page 720
- "Testing the proxy servlet" on page 730

HTTP proxy servlet; web services clients component

The HTTP proxy servlet requires a number of components such as message flows, a proxy servlet, a servlet container, web addresses, WebSphere Message Broker, and web services clients. Ensure that you are familiar with the web services clients component.

web services clients are applications that send and receive SOAP (or simple HTTP) requests and responses to and from web services implemented in message flows running in a broker.

Before you install and test the HTTP proxy servlet, ensure you understand the following concepts:

- "HTTP proxy servlet overview" on page 709
- "HTTP traffic handling in WebSphere Message Broker" on page 710
- "HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 711
- "HTTP proxy servlet; descriptions of required components" on page 715

When you have gained an understanding of the proxy servlet concept, read the following topics to help you install and test the HTTP proxy servlet:

- "Installing the proxy servlet"
- "Testing the proxy servlet" on page 730

Installing the proxy servlet

The installation of the proxy servlet is made up of a number of tasks. Use this topic to complete each of the tasks to then move onto testing the proxy servlet.

Before you install the HTTP proxy servlet, ensure you understand the following concepts:

- "HTTP traffic handling in WebSphere Message Broker" on page 710
- "HTTP traffic handling by using the proxy servlet in an external web servlet container" on page 711
- "HTTP proxy servlet; descriptions of required components" on page 715

The proxy servlet `proxyservlet.war` (WAR) file is part of the runtime environment in WebSphere Message Broker Version 6.1 Fix Pack 3 (6.1.0.3) and above, and can be found in the following directory:

WMB61_runtime_installation_path`/tools`. Where *WMB61_runtime_installation_path* specifies the name of your runtime installation directory.

Supported operating systems

The proxy servlet runs on any operating system that is supported by the servlet container. The documentation in this section describes the installation on Windows XP but the proxy servlet can be installed on other operating systems.

Prerequisites

The proxy servlet requires:

- WebSphere Message Broker Version 6.1 Fix Pack 3 with iFixes, or later.
- WebSphere MQ Version 6.0 Fix Pack 4 (WebSphere MQ classes for Java), or later.
- A servlet container and web server, such as WebSphere Application Server Version 6, or Apache Tomcat Version 6.

To install the proxy servlet:

1. Install and customize a web servlet container, such as WebSphere Application Server or Apache Tomcat. See "Installing and customizing a web servlet container for the proxy servlet" for more information.
2. Configure the proxy servlet with the initialization parameters that are used by the proxy servlet. See "Configuring the proxy servlet" on page 722 for more information.
3. Enable the WebSphere MQ listener on each execution group where message flows with SOAP nodes are deployed so that the proxy servlet can access the SOAPInput and SOAPReply nodes. See "Enabling the WebSphere MQ listener for SOAP nodes for the proxy servlet to access" on page 729 for more information.
4. Deploying the proxy servlet in the web servlet container. See "Deploying the proxy servlet in the web servlet container" on page 729 for more information.

You are now ready to test the proxy servlet. For information on how to complete this task, see "Testing the proxy servlet" on page 730.

Installing and customizing a web servlet container for the proxy servlet

Download, install, and customize a web servlet container, such as WebSphere Application Server or Apache Tomcat, for the proxy servlet to use to receive HTTP requests from web services client applications.

Before you install the servlet container, you must have completed the following tasks:

- Installed WebSphere Message Broker.
- If you are planning to install Apache Tomcat V6 as your web servlet, you must have installed a Java SDK that is compatible with Apache Tomcat V6.

The documentation in this section is based on an Apache Tomcat Version 6 installation, however this procedure can be used to guide you through the installation of a different web servlet container because the procedures are similar. This information describes the Apache Tomcat V6 installation on Windows XP and assumes WebSphere Message Broker is installed and running on the same server.

Windows

1. Download `apache-tomcat-6.1.18.exe` to your local directory from the Apache Tomcat 6 Downloads website at: Apache Tomcat 6 download page
2. Double-click `apache-tomcat-6.1.18.exe` in your local directory to run the installer.
   a. Accept the license agreement.
   b. Select the required Apache Tomcat components and features to install.
   c. Select the destination directory for the Apache Tomcat installation. For example, `C:\Tomcat6`.
   d. Enter an HTTP listener port. For example, 8181.
   e. Type an administrator user ID and password. For example, *admin*.
   f. Browse and select a path for the J2SE 5.0 Java Runtime Environment (JRE). For example, `C:\Program Files\IBM\Java50\jre`.
   g. Click **Install** to start the installation.
3. Restart Windows. Apache Tomcat is now running as a Windows Service.
4. Open a web browser, such as Internet Explorer, and enter web address http://localhost:8181. Apache Tomcats home page is displayed.

5. Find file `catalina.properties` in your local directory. For example, *<tomcat_installation_path>*`/conf`. Where *<tomcat_installation_path>* specifies the name of your Apache Tomcat installation directory.
6. Edit `catalina.properties` and change **shared.loader=** to **shared.loader= ${catalina.home}/shared/lib,${catalina.home}/shared/lib/*.jar**.
7. Create a directory called *<tomcat_installation_path>*`/shared/lib`.
8. The proxy servlet uses WebSphere MQ to communicate with WebSphere Message Broker. It requires the following JAR files to exist in the Apache Tomcat shared loader directory: Copy `com.ibm.mq.jar` and `connector.jar` from *<WebSphere_MQ_installation_path>*`\Java\lib` to *<tomcat_installation_path>*`\shared\lib`.

You have downloaded, installed, and customized Apache Tomcat V6 ready for use by the proxy servlet.

You must now configure the proxy servlet, see:

- "Configuring the proxy servlet"

Configuring the proxy servlet

Configure the proxy servlet with the initialization parameters that are used by the proxy servlet. These parameters need to be configured for the broker environment that the proxy servlet is connecting to before the proxy servlet can be deployed to the servlet container.

Before you configure the proxy servlet, you must have completed the following task:

- "Installing and customizing a web servlet container for the proxy servlet" on page 721

The configuration of the proxy servlet is done by editing the Web Deployment Descriptor `web.xml` file that is packaged in the compressed `proxyservlet.war` (WAR) file. The WAR file can be found in the following directory:

<WMB_runtime_install_path>`/tools`.

You can either use the WebSphere Message Broker toolkit to edit the Web Deployment Descriptor `web.xml` file, or you can extract the `proxyservlet.war` file, find the `web.xml` file and edit it using an appropriate editor, such as Notepad. Both procedures are described in this section:

1. To configure the Web Deployment Descriptor `web.xml` file using the WebSphere Message Broker toolkit, start the toolkit, and switch to the J2EE perspective.
2. Click **File** → **Import**, expand the **Web** section, select **WAR file** in the list, and click **Next**.
3. Click **Browse** to find the WAR file in *<WMB_runtime_install_path>*`/tools`. Where *<WMB_runtime_install_path>* specifies the name of your runtime installation path. For example, in `C:\Program Files\IBM\MQSI\7.0\tools\proxyservlet.war`.
4. Set the name of the Web Project to `proxyservlet`.
5. Click **Finish**. The proxy servlet is now ready for configuring using the J2EE perspective.
6. Expand **proxyservlet** in the Project Explorer view and double-click Deployment Descriptor to view the Web Deployment Descriptor.

7. Find the **Servlets and JSPs** section in the Web Deployment Descriptor, and click the servlet link called **WBIMBServlet** to display the servlet web address mappings and initialization parameters.
8. Click the Source tab, which is found at the bottom of the Web Deployment Descriptor view. You might need to click **>>** (Show List) to see the Source tab option. The source of the Web Deployment Descriptor `web.xml` displays the proxy servlet parameters.
9. Edit the proxy servlet parameters with initialization parameters specified at: "Proxy servlet configuration parameters"
10. When the configuration is complete, save the changes to the Deployment Descriptor `web.xml` file by pressing **Ctrl S**.
11. Export the configured proxy servlet ready for deployment to Tomcat. Click **File → Export**, expand the **Web** section, select **WAR file** in the list, and click **Next**.
12. Click **Browse**, specify a location for the configured WAR file, enter WAR file name `HTTPVSR1BKproxyserlet.war`, and click **Save**.
13. Enter the Web module name `proxyservlet`, and click **Finish**. You have now configured the proxy servlet using the WebSphere Message Broker toolkit.

To configure the Web Deployment Descriptor `web.xml` file directly, find and extract the `proxyservlet.war` file, find the `web.xml` file in the extracted contents, right-click the `web.xml` file, and select an appropriate editor, such as Notepad to edit the `web.xml` file with initialization parameters specified at: "Proxy servlet configuration parameters."

You have now configured the proxy servlet with the initialization parameters.

To access to the SOAPInput and SOAPReply nodes from the proxy servlet you must now enable the WebSphere MQ listener on each execution group where message flows with SOAP nodes are deployed, see:

- "Enabling the WebSphere MQ listener for SOAP nodes for the proxy servlet to access" on page 729

Proxy servlet configuration parameters:

The proxy servlet `web.xml` file must be configured with the initialization parameters specified in this topic for the broker environment that the proxy servlet is connecting to before the proxy servlet can be deployed to the servlet container.

General options:

Parameter name	Default value	Description
brokerName	* (auto-detect from config data for broker)	*broker name* or "*" Use this parameter to set the name used for error messages; the value is auto-detected if set to "*". Set a value if several brokers are being proxied, and a single name is required for error messages.

Parameter name	Default value	Description
forwardingNodes	http	*http*, *soap*, or *both* Whether to forward traffic to the HTTP nodes, the SOAP nodes, or both. Use *both* with caution, because the merging of Web address selectors might lead to unexpected results in conflicting cases; for example, if an HTTP node and a SOAP node both use `/fnerble` as a Web address selector.
configFilePath	`/var/mqsi/components/WBRK61_DEFAULT_BROKER/config/wsplugin6.conf`	*full path to config file* If the proxied broker is local, set this parameter to the `wsplugin6.conf` file (for HTTP nodes) or the `soapplugin6.conf` (for SOAP nodes) for the broker. This file is used only when the parameter **useQueueManagerDataInsteadOfConfigFile** is set to blank. The configuration file can be used only when the proxy servlet is running on the same server as the broker, and it has access to the file. In Windows, the file is stored in `C:\`*`install_dir`*`\config\wsplugin.conf` or `C:\Documents and Settings\All Users\Application Data\IBM\MQSI\components\`*`broker_ name`*`\config\wsplugin6.conf`. On Linux and UNIX, the file is stored in `/var/mqsi/config/wsplugin.conf` or in `/var/mqsi/components/`*`broker_ name`*`/config/wsplugin6.conf`.
useFastpathBindingsConnection	false	*true* or *false* Causes the servlet to connect in fastpath mode, if using a local queue manager.
traceFileName		*full path to trace file* Specify the location and name of the trace file. If this parameter is not specified the trace is sent to stdout.
turnTraceOn	0	*0*, *1*, or *2* Set 0 for no trace, 1 for normal trace, or 2 for debug trace.

Information options:

Parameter name	Default value	Description
enableStatusPage (WebSphere Message Broker V6.1 FP 4 or later)	false	*true* or *false* Switches display of the status page. When true, the page is visible at `http://hostname:port/proxy_context/messagebroker/httpproxy/statuspage`
enableInfoHeaders (WebSphere Message Broker V6.1 FP 4 or later)	false	*true* or *false* Causes the servlet to add extra headers in the response. These headers are: X-WMB-Broker-Name X-WMB-QM-Name X-WMB-MQ-URL-CorrelId and contain details of the configuration used for that message.

ReplyToQ and QMgr options:

Parameter name	Default value	Description
useClusterMode	false	*true* or *false* Set to true if the servlet is required to put reply-to queue and queue manager information in the MQMD of sent messages to enable the broker to respond to the correct queue manager in a cluster.
clusterModeQueueManagerName	SOME_OTHER_ QUEUE_MANAGER	*queue manager name* Queue manager name for initial MQCONN and ReplyToQMgr.
clusterModeReplyToQ	OUR.REPLYTO.QUEUE	*reply queue name* Queue name on which to listen.

MQ connection options:

Parameter name	Default value	Description
useClientMode	false	*true* or *false* Use WebSphere MQ client (true) or bindings connection (false). Normally, **useQueueManagerDataInsteadOfConfigFile** would also be set to the broker queue manager if this parameter is set to true.
clientModeHostname	localhost	*hostname or IP address* Hostname or IP for the Queue Manager.
clientModeChannelName	SYSTEM.DEF. SVRCONN	*WebSphere MQ SVRCONN channel name* The name of the WebSphere MQ SVRCONN to use.

Parameter name	Default value	Description
clientModePortNumber	1414	*port number* WebSphere MQ listener port number.
clientModeConnectRetryCount (WebSphere Message Broker V6.1 FP 4 or later)	1	*integer* Number of times to retry the WebSphere MQ connect call. Use this parameter in cases where a network dispatcher or load balancer is being used to distribute work to a set of queue managers and one fails. A new connect might fail the first time, but succeed the second time. The retry count must be set to a high number to provide the greatest chance of success.
useQueueManagerDataInsteadOfConfigFile		*queue manager name*, "*", or *blank* Queue manager name, '*' (remote proxy), or blank for none (local proxy). This option causes the servlet to read Web address data from a queue, and avoid the need for a config file to be accessible from the servlet.
sleepBeforeGet	0	*time in seconds* Sleep time in seconds. This value causes the servlet to wait before issuing an MQGET for a response message from the broker.
disconnectBeforeSleep	true	*true* or *false* To release WebSphere MQ handle while sleeping. Useful for keeping the number of simultaneous WebSphere MQ connections down.
reconnectActiveLinksAge (WebSphere Message Broker V6.1 FP 4 or later)	-1	*time in seconds*, 0, or -1 If set to a number greater than zero, this parameter causes WebSphere MQ connections to be disconnected and reconnected if they have been inactive, because of low traffic volumes, for more than the specified number of seconds. Setting this to -1 prevents this reconnection. Setting it to 0 causes all connections to be used once only. This parameter is of most use if the connection to WebSphere MQ goes through a firewall that closes connections after a period of inactivity. Setting this parameter to a value less than the firewall timeout might prevent clients from getting WebSphere MQ 2009 (connection broken) errors.

Parameter name	Default value	Description
testConnectionBeforeReuse (WebSphere Message Broker V6.1 FP 4 or later)	false	*true* or *false* If set to true, the servlet attempts an MQINQ before doing the MQPUT of the HTTP data message. All problems with a cached WebSphere MQ client connection are detected at that point, and a new connection is established for the MQPUT of the actual data (and MQGET of the response). This parameter causes significant extra network traffic, and must be used only if problems have been seen with dropped connections, which are usually seen as WebSphere MQ 2009 errors, indicating connection broken.
maximumConnectionAge (WebSphere Message Broker V6.1 FP 6 or later)	-1	*time in seconds*, 0, or -1 If set to a number greater than zero, this parameter causes WebSphere MQ connections to be disconnected and reconnected if they are older than the specified number of seconds. Setting this parameter to -1 prevents these reconnections; setting this parameter to 0 causes all connections to be used only once. This parameter is of most use, if the frequent changes to the WebSphere MQ connection parameters are expected due to redeploys of the WebSphere Message Broker flows and you require the *ProxyServlet* to reflect these changes within the specified number of seconds.

You can define one or more mappings that are supported by the proxy servlet. These mappings are used by the servlet container to filter Web address requests before executing the correct instance of the proxy servlet.

The mappings are the */node_url_path* paths described in "HTTP proxy servlet; web addresses component" on page 717 (*/url_path=/context_root/node_url_path*).

You can define a */node_url_path* equal to "/*" to accept Web address paths similar to this example: /HTTPMyBrkServletProxy/*your_value*.

Example configuration scenarios:

In each of the following scenarios, **forwardingNodes** must be set for the HTTP and SOAP nodes:

- Scenario 1: The Web servlet container is on the same server as WebSphere Message Broker:

  In this configuration example, you must set **useClientMode** and **useQueueManagerDataInsteadOfConfigFile** to false, and the **configFile**

parameter must point to a valid file. The servlet attempts to connect to the local queue manager for the broker after reading the queue manager name from the config file.

- Scenario 2: The Web servlet container is on a different server to WebSphere Message Broker with an WebSphere MQ client link to the broker queue manager:

  In this configuration example, you must set **useClientMode** to true, **useQueueManagerDataInsteadOfConfigFile** to "*", or the broker queue manager name, and **clientModeHostname**, **clientModeChannelName**, and **clientModePortNumber** to the correct values defined in this section. The servlet attempts to connect to the remote queue manager for the broker, reading the required node configuration data from the broker from the WebSphere MQ client connection.

  You can copy the config file from the broker server to the Web servlet container server, and then use **configFile**, after ensuring that **useQueueManagerDataInsteadOfConfigFile** is set to blank to force reading from the config file. However, you must copy the config file every time it is changed by the broker.

- Scenario 3: The Web container is on a different server to WebSphere Message Broker with its own queue manager and a WebSphere MQ channel link to the broker queue manager:

  This configuration example is similar to scenario 1 in that client mode is not used, but you must set **useClusterMode** to true, **clusterModeQueueManagerName** to the queue manager of the Web servlet container, and **clusterModeReplyToQ** to a queue that exists on that queue manager.

  The servlet tries to open the queue SYSTEM.BROKER.WS.INPUT on the specified queue manager by using the queue manager name from the config file. Therefore, you must set up channels and transmit queues beforehand, to ensure the messages arrive on the broker queue manager.

  You must copy the config files from the broker server in this scenario.

- Scenario 4: This scenario is the same as scenario 2, but uses a network load-balancer for distributing work for several brokers:

  The configuration can be the same as scenario 2, with the network load-balancer IP address taking the place of the broker server. In general, config files cannot be used, because there are several brokers behind one virtual IP address, and each one has a different config file. The servlet loads information on a per-connection basis, and uses the correct configuration information for each broker.

  Because failover is often one of the reasons for this configuration, the following extra options can be useful:

  Set **clientModeConnectRetryCount** to ensure that a single failed server does not cause intermittent errors, even if the load-balancer does simple round connection dispatching. Setting the parameter to a high enough value to attempt to connect to all the brokers prevents problems in these cases, and the servlet uses the first available broker.

  Use **reconnectActiveLinksAge** to avoid reusing old connections that might have been discarded by firewalls in between the servlet and the load-balancer (or the load-balancer and the brokers). Set this parameter to a value less than the firewall timeout, to ensure connections are used only when they are valid.

  Use **testConnectionBeforeReuse** as an alternative way to handle dropped WebSphere MQ links between the Web servlet container and broker queue managers. This option causes an MQINQ to be performed before attempting to put any data to the broker. If the MQINQ fails, a new connection is established,

and the data is sent over the new connection. Because configuration adds another operation to the MQPUT and MQGET, it results in a significant overhead for every message; use this option only if no alternative options are available.

To finish completing the proxy servlet configuration, see "Configuring the proxy servlet" on page 722.

Enabling the WebSphere MQ listener for SOAP nodes for the proxy servlet to access

Enable the WebSphere MQ listener on each execution group where message flows with SOAP nodes are deployed so that the proxy servlet can access the SOAPInput and SOAPReply nodes.

Before you start:

Before you enable the WebSphere MQ listener, you must have completed the following tasks:

- "Installing and customizing a web servlet container for the proxy servlet" on page 721
- "Configuring the proxy servlet" on page 722

1. Open a command window that is configured for your environment and enter the following command:

   ```
   mqsichangeproperties <broker_name> -e <execution_group_name> -o HTTPConnector
   -n enableMQListener -v true
   ```

   For example:

   ```
   mqsichangeproperties VSR1BK -e default -o HTTPConnector -n enableMQListener -v true
   ```

2. Stop and start the broker.
3. Enter the following command to verify the property value:

   ```
   mqsireportproperties <broker_name> -e <execution_group_name> -o HTTPConnector
   -n enableMQListener
   ```

   For example:

   ```
   mqsireportproperties VSR1BK -e default -o HTTPConnector –n enableMQListener
   ```

4. Repeat these steps for all execution groups where message flows with SOAP nodes are deployed.

You have enabled the WebSphere MQ listener for all execution groups where message flows with SOAP nodes are deployed.

You are now ready to deploy the proxy servlet in the Web servlet container. For information about how to complete this task, see "Deploying the proxy servlet in the web servlet container."

Deploying the proxy servlet in the web servlet container

Load and install the proxy servlet file in the web servlet container, such as WebSphere Application Server or Apache Tomcat.

The documentation in this section is based on deploying the proxy servlet in Apache Tomcat Version 6 on Windows XP and assumes WebSphere Message Broker is installed and running on the same server.

Before you deploy the proxy servlet in Apache Tomcat, you must have completed the following tasks:

- "Installing and customizing a web servlet container for the proxy servlet" on page 721
- "Configuring the proxy servlet" on page 722
- "Enabling the WebSphere MQ listener for SOAP nodes for the proxy servlet to access" on page 729

1. Open a web browser and enter web address http://localhost:8181 to open the Tomcat home page.
2. Click on **Tomcat Manager** and enter the *admin* user ID and password.
3. Scroll down to the section **WAR file to deploy** and click **Browse** and search for the `HTTPVSR1BKproxyservlet.war` file.
4. Click **Deploy**. The <*context_root*> associated to the proxy servlet is the same as the WAR filename. It is possible to rename the `.war` file before deploying it to Tomcat to have a different <*context_root*>. See "HTTP proxy servlet; web addresses component" on page 717 for a description of the <*context_root*>.
5. Click **/HTTPVSR1BKproxyservlet** on the Applications view to test if the servlet is active. If the proxy servlet trace is enabled, the trace file will show that the servlet has received the request and rejected it.

The proxy servlet is now ready to be tested with a message broker receiving HTTP (not HTTPS) requests and passing them to a message flow. For information on how to complete this task see:

- "Testing the proxy servlet"

Testing the proxy servlet

Test the proxy servlet with a message broker receiving HTTP requests and passing them to a message flow.

To test the proxy servlet, use an existing web services client application or write your own SSL test client application using Java.

Before you test the proxy servlet, you must have completed the following tasks:

- "Installing and customizing a web servlet container for the proxy servlet" on page 721
- "Configuring the proxy servlet" on page 722
- "Enabling the WebSphere MQ listener for SOAP nodes for the proxy servlet to access" on page 729
- "Deploying the proxy servlet in the web servlet container" on page 729

1. Install a client application that can send HTTP or SSL (HTTPS) requests. There are several options that can be used, such as:
   - Nettool: An open source graphical utility available from the Internet at: Nettool downloads Web site. This tool allows to send HTTP requests or to tunnel a request.
   - OpenSSL: Another open source tool available from the Internet at: OpenSSL downloads Web site that can be used to send SSL (HTTPS) requests to the web servlet container.
   - A broker message flow that has a HTTPRequest or SOAPRequest node can generate and send HTTP requests to a HTTP listener.
   - A web browser using web pages or Java Server Pages (JSP) that can send HTTP POST requests. Most web browsers support HTTP and SSL (HTTPS).
   - A client application that sends requests using HTTP, or (SSL) HTTPS, or both.
2. Configure the proxy servlet to access the message broker in the `web.xml` file.

3. Configure a message flow with HTTP and SOAP input and reply nodes. The message flow receives the messages from the proxy servlet and if a HTTP and SOAP reply node is configured, responses are sent back to the proxy servlet.

You have now finished testing the proxy servlet.

Part 2. Working with Web services

Working with Web services

Use WebSphere Message Broker nodes and services to connect to Web services.

A Web service is a software system designed to support interoperable computer-to-computer interaction over a network. It has an interface described by an XML-based specification; specifically, the Web Service Definition Language, or WSDL.

Web services fulfill a specific task or a set of tasks. A Web service is described using a standard, formal XML notion, called its service description, that provides all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols, and location.

The nature of the interface hides the implementation details of the service, so that it can be used independently of the hardware or software platform on which it is implemented. The interface is also independent of the programming language in which it is written.

This interface handles Web service-based applications as loosely coupled, component-oriented, cross-technology implementations. Web services can be used alone, or with other Web services, to carry out a complex aggregation or a business transaction.

The following topics describe how to work with Web services:

- "WebSphere Message Broker and Web services"
- "What is SOAP?" on page 737
- "What is WSDL?" on page 744
- "What is SOAP MTOM?" on page 748
- "WS-Addressing" on page 749
- "WS-Security" on page 759
- "WebSphere Service Registry and Repository" on page 781
- "External standards" on page 807
- "Message flows for Web services" on page 815

WebSphere Message Broker supplies a Java servlet that you can use in an external Web servlet container such as IBM WebSphere Application Server or Apache Tomcat, to receive HTTP requests from Web services client applications. The HTTP proxy servlet is described in "HTTP proxy servlet overview" on page 709.

WebSphere Message Broker and Web services

A WebSphere Message Broker application can participate in a Web services environment as a service requester, as a service provider, or both.

The SOAP domain supports these formats:

- Common Web services message formats SOAP 1.1, SOAP 1.2, SOAP with Attachments (SwA), and MTOM.
- Consistent SOAP logical tree format, which is independent of the exact message format.

- WS-Addressing and WS-Security standards.

The following nodes are provided for use in the SOAP domain:
- "SOAPInput node" on page 1194
- "SOAPReply node" on page 1202
- "SOAPRequest node" on page 1204
- "SOAPAsyncRequest node" on page 1172
- "SOAPAsyncResponse node" on page 1181
- "SOAPEnvelope node" on page 1186
- "SOAPExtract node" on page 1189

Use the SOAP nodes and SOAP domain where possible; see "Web services: when to use SOAP or HTTP nodes" on page 737.

Web services support conforms to the following open standards:
- SOAP 1.1 and 1.2
- SOAP Messages with Attachments
- MTOM
- HTTP 1.1
- WSDL 1.1
- WS-Addressing (new SOAP domain only)
- WS-Security (new SOAP domain only)

WSDL is also validated against the WS-I Basic Profile Version 1.1. Conformance to the guidelines in this specification improves interoperability with other applications.

For more information about how a WebSphere Message Broker application can participate in a Web services environment, see the WebSphere Message Broker Web page on developerWorks.

What is a Web service?

A Web service is defined by the World Wide Web Consortium (W3C) as a software system designed to support interoperable machine-to-machine interaction over a network.

A Web service fulfills a specific task or a set of tasks, and is described by a service description in a standard XML notation called Web Services Description Language (WSDL). The service description provides all of the details necessary to interact with the service, including message formats (that detail the operations), transport protocols, and location.

Other systems use SOAP messages to interact with the Web service, typically by using HTTP with an XML serialization in conjunction with other Web-related standards.

The WSDL interface hides the details of how the service is implemented, and the service can be used independently of the hardware or software platform on which it is implemented, and independently of the programming language in which it is written.

Applications that are based on Web services are loosely-coupled, component-oriented, cross-technology implementations. Web services can be used alone, or in conjunction with other Web services to carry out a complex aggregation or a business transaction.

Web services: when to use SOAP or HTTP nodes

HTTP and SOAP nodes can both be used to interact with Web services. Typically you use SOAP nodes when working with SOAP-based Web services.

For SOAP-based Web services, several advantages exist if you use the SOAP nodes and the SOAP message domain instead of the HTTP transport nodes and XMLNSC message domain.

- Support for WS-Addressing, WS-Security and SOAP headers.
- A common SOAP logical tree format, independent of the bitstream format.
- Runtime checking against WSDL.
- Automatic processing of SOAP with Attachments (SwA).
- Automatic processing of Message Transmission Optimization Mechanism (MTOM).

Although the HTTP nodes can process SwA messages, you must use the MIME message domain and design your flow to handle the attachments explicitly, and use custom logic to extract and parse the SOAP.

Cases where it might be better to use HTTP nodes include:

- Message flows in which a single request node handles multiple SOAP request and responses from more than one WSDL.
- Message flows that interact with Web services that use different standards, such as REST or XML-RPC.
- Message flow that never use WS-Addressing, WS-Security, SwA, or MTOM.

What is SOAP?

SOAP is an XML message format used in Web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific Web service is described by a WSDL definition.

There are two versions of SOAP in common use: SOAP 1.1 and SOAP 1.2. Both are supported in WebSphere Message Broker. SOAP is defined in the following documents issued by World Wide Web Consortium (W3C):

- Simple Object Access Protocol (SOAP) 1.1 (W3C note).
- SOAP Version 1.2 Part 0: Primer (W3C recommendation).
- SOAP Version 1.2 Part 1: Messaging Framework (W3C recommendation).
- SOAP Version 1.2 Part 2: Adjuncts (W3C recommendation).

Support for SOAP in WebSphere Message Broker includes:

- SOAP parser and domain. See “SOAP parser and domain” on page 96.
- SOAP nodes to send and receive messages in SOAP format.
- IBM supplied message definitions for SOAP 1.1 and SOAP 1.2. These message definitions support validation, ESQL content assist, and the creation of message maps for use with SOAP messages, in the SOAP and other XML domains. See IBM supplied messages that you can import.

WSDL validation in WebSphere Message Broker refers to the WS-I Basic Profile. For more information, see the WS-I, and in particular the WS-I Basic Profile document:

- http://www.ws-i.org/
- http://www.ws-i.org/deliverables

The structure of a SOAP message

A SOAP message is encoded as an XML document, consisting of an <Envelope> element, which contains an optional <Header> element, and a mandatory <Body> element. The <Fault> element, contained in <Body>, is used for reporting errors.

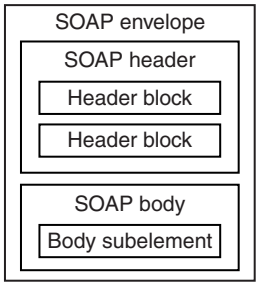
The SOAP envelope
<Envelope> is the root element in every SOAP message, and contains two child elements, an optional <Header> element, and a mandatory <Body> element.

The SOAP header
<Header> is an optional subelement of the SOAP envelope, and is used to pass application-related information that is to be processed by SOAP nodes along the message path; see "The SOAP header" on page 739.

The SOAP body
<Body> is a mandatory subelement of the SOAP envelope, which contains information intended for the ultimate recipient of the message; see "The SOAP body" on page 740.

The SOAP fault
<Fault> is a subelement of the SOAP body, which is used for reporting errors; see "The SOAP fault" on page 741.

XML elements in <Header> and <Body> are defined by the applications that make use of them, although the SOAP specification imposes some constraints on their structure. The following diagram shows the structure of a SOAP message.



The following code is an example of a SOAP message that contains header blocks (the <m:reservation> and <n:passenger> elements) and a body (containing the <p:itinterary> element).

```
<?xml version='1.0' Encoding='UTF-8' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
  env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2007-11-29T13:20:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
  env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
   <n:name>Fred Bloggs</n:name>
```

```
  </n:passenger>
 </env:Header>
 <env:Body>
  <p:itinerary xmlns:p="http://travelcompany.example.org/reservation/travel">
   <p:departure>
     <p:departing>New York</p:departing>
     <p:arriving>Los Angeles</p:arriving>
     <p:departureDate>2007-12-14</p:departureDate>
     <p:departureTime>late afternoon</p:departureTime>
     <p:seatPreference>aisle</p:seatPreference>
   </p:departure>
   <p:return>
     <p:departing>Los Angeles</p:departing>
     <p:arriving>New York</p:arriving>
     <p:departureDate>2007-12-20</p:departureDate>
     <p:departureTime>mid-morning</p:departureTime>
     <p:seatPreference></p:seatPreference>
   </p:return>
  </p:itinerary>
 </env:Body>
</env:Envelope>
```

The SOAP header

The SOAP header (the <Header> element) is an optional sub-element of the SOAP envelope, and is used to pass application-related information that is processed by SOAP nodes along the message flow.

The immediate child elements of the header are called *header blocks*. A header block is an application-defined XML element, and represents a logical grouping of data which can be targeted at SOAP nodes that might be encountered in the path of a message from a sender to an ultimate receiver.

SOAP header blocks can be processed by SOAP intermediary nodes, and by the ultimate SOAP receiver node. However, in a real application, not every node processes every header block. Each node is typically designed to process particular header blocks, and each header block is processed by particular nodes.

The SOAP header enables you to add features to a SOAP message in a decentralized manner without prior agreement between the communicating parties. SOAP defines some attributes that can be used to indicate what can deal with a feature and whether it is optional or mandatory. Such control information includes, for example, passing directives or contextual information related to the processing of the message. This control information enables a SOAP message to be extended in an application-specific manner.

Although the header blocks are application-defined, SOAP-defined attributes on the header blocks indicate how the header blocks must be processed by the SOAP nodes. SOAP-defined attributes include:

encodingStyle
: Indicates the rules used to encode the parts of a SOAP message. SOAP defines a narrower set of rules for encoding data than the flexible encoding that XML enables.

actor (SOAP 1.1) or role (SOAP 1.2)
: In SOAP 1.2, the role attribute specifies whether a particular node will operate on a message. If the role specified for the node matches the role attribute of the header block, the node processes the header. If the roles do not match, the node does not process the header block. In SOAP 1.1, the actor attribute performs the same function.

Roles can be defined by the application, and are designated by a URI. For example, `http://example.com/Log` might designate the role of a node which performs logging. Header blocks that are processed by this node specify `env:role="http://example.com/Log"` (where the namespace prefix `env` is associated with the SOAP namespace name of `http://www.w3.org/2003/05/soap-envelope`).

The SOAP 1.2 specification defines three standard roles in addition to those which are defined by the application:

`http://www.w3.org/2003/05/soap-envelope/none`
None of the SOAP nodes on the message path should process the header block directly. Header blocks with this role can be used to carry data that is required for processing of other SOAP header blocks.

`http://www.w3.org/2003/05/soap-envelope/next`
All SOAP nodes on the message path are expected to examine the header block (provided that the header has not been removed by a node earlier in the message path).

`http://www.w3.org/2003/05/soap-envelope/ultimateReceiver`
Only the ultimate receiver node is expected to examine the header block.

mustUnderstand
This attribute is used to ensure that SOAP nodes do not ignore header blocks which are important to the overall purpose of the application. If a SOAP node determines, by using the **role** or **actor** attribute, that it should process a header block, the action taken depends on the value of the **mustUnderstand** attribute.

- `1` (SOAP 1.1) or `true` (SOAP 1.2): The node must either process the header block in a manner consistent with its specification, or not at all (and throw a fault).
- `0` (SOAP 1.1) or `false` (SOAP 1.2): The node is not obliged to process the header block.

In effect, the **mustUnderstand** attribute indicates whether processing of the header block is mandatory or optional.

relay (SOAP 1.2 only)
When a SOAP intermediary node processes a header block, the SOAP intermediary node removes the header block from the SOAP message. By default, the SOAP intermediary node also removes all header blocks that it ignored (because the **mustUnderstand** attribute had a value of `false`). However, when the relay attribute is specified with a value of `true`, the SOAP intermediary node retains the unprocessed header block in the message.

The SOAP body

The SOAP body (the <Body> element) is a mandatory sub-element of the SOAP envelope, which contains information intended for the ultimate recipient of the message.

The body element and its associated child elements are used to exchange information between the initial SOAP sender and the ultimate SOAP receiver. SOAP defines one child element for the body: the `<Fault>` element, which is used for reporting errors. Other elements in the body are defined by the Web service that uses them.

The SOAP fault

The SOAP fault (the <Fault> element) is a sub-element of the SOAP body, which is used for reporting errors.

If present, the SOAP fault element must appear as a body entry and must not appear more than once in a body element. The sub-elements of the SOAP fault element are different in SOAP 1.1 and SOAP 1.2.

SOAP 1.1

In SOAP 1.1, the SOAP fault contains the following sub-elements:

<faultcode>
The<faultcode> element is a mandatory element in the <Fault> element. It provides information about the fault in a form that can be processed by software. SOAP defines a small set of SOAP fault codes covering basic SOAP faults, and this set can be extended by applications.

<faultstring>
The <faultstring> element is a mandatory element in the <Fault> element. It provides information about the fault in a form intended for a human reader.

<faultactor>
The <faultactor> element contains the URI of the SOAP node that generated the fault. A SOAP node that is not the ultimate SOAP receiver must include the <faultactor> element when it creates a fault; an ultimate SOAP receiver is not obliged to include this element, but might do so.

<detail>
The <detail> element carries application-specific error information related to the <Body> element. It must be present if the contents of the <Body> element were not successfully processed. The <detail> element must not be used to carry information about error information belonging to header entries. Detailed error information belonging to header entries must be carried in header entries.

SOAP 1.2

In SOAP 1.2, the SOAP fault contains the following sub-elements:

<Code> The <Code> element is a mandatory element in the <Fault> element. It provides information about the fault in a form that can be processed by software. It contains a <Value> element and an optional <Subcode> element.

<Reason>
The <Reason> element is a mandatory element in the <Fault> element. It provides information about the fault in a form intended for a human reader. The <Reason> element contains one or more <Text> elements, each of which contains information about the fault in a different language.

<Node> The <Node> element contains the URI of the SOAP node that generated the fault. A SOAP node that is not the ultimate SOAP receiver must include the <Node> element when it creates a fault; an ultimate SOAP receiver is not obliged to include this element, but might do so.

<Role> The <Role> element contains a URI that identifies the role in which the node was operating at the point the fault occurred.

<Detail>

The <Detail> element is an optional element, which contains application-specific error information related to the SOAP fault codes describing the fault. The presence of the <Detail> element has no significance as to which parts of the faulty SOAP message were processed.

SOAP nodes

The SOAP nodes act as points in the flow where Web service processing is configured and applied. Properties on the SOAP nodes control the processing carried out and can be configured by supplying a WSDL definition, or by manually configuring properties, or both.

SOAP nodes

- The SOAPInput and SOAPReply nodes are used in a message flow which implements a Web service. These SOAP nodes are used to construct a message flow that implements a Web service provider. The SOAPInput node listens for incoming Web service requests, and the SOAPReply sends responses back to the client; see “SOAPInput node” on page 1194 and “SOAPReply node” on page 1202.
- The SOAPRequest node is used in a message flow to call a Web service provider synchronously. Calling a Web service synchronously means that the node sends a Web service request and waits, blocking the message flow, for the associated Web service response to be received before the message flow continues; see “SOAPRequest node” on page 1204.
- The SOAPAsyncRequest and SOAPAsyncResponse nodes are used to construct a message flow (or pair of flows) which calls a Web service asynchronously. Calling a Web service asynchronously means that the SOAPAsyncRequest node sends a Web service request, but the request does not block the message flow by waiting for the associated Web service response to be received because the Web service response is received at the SOAPAsyncResponse node, which is in a separate flow. The Node Correlator identifies the logical pairing of the responses against the original requests. Multiple requests can, therefore, be handled in parallel; see “SOAPAsyncRequest node” on page 1172 and “SOAPAsyncResponse node” on page 1181.
- You can work on the payload of the SOAP body using the SOAPExtract and SOAPEnvelope nodes. The SOAPExtract node can interoperate with the SOAP domain. The SOAP nodes do not require the SOAPEnvelope node, because they can directly handle non-SOAP messages (and look at the local environment) but the SOAPEnvelope node is still required for the HTTP nodes. See “SOAPExtract node” on page 1189 and “SOAPEnvelope node” on page 1186.

The W3C SOAP specification refers to "SOAP nodes" meaning a unit of application logic (see Web Services Glossary). Typically, references to "SOAP nodes" in the WebSphere Message Broker Information Center are referring to WebSphere Message Broker SOAP nodes.

SOAP applications

SOAP is an XML based language defined by the World Wide Web Consortium (W3C) for sending data between applications. SOAP is transport and platform neutral.

SOAP message

A SOAP message comprises an envelope containing:

- An optional header (containing one or more header blocks).
- A mandatory body.

The content of the header and body is typically defined by WSDL.

SOAP style

SOAP defines two types of style:

RPC The SOAP body corresponds to a method call.

document
The SOAP body is typically a coarser-grained XML document and is defined explicitly by XML Schema.

SOAP encodings

SOAP defines two types of encoding:

SOAP encoding
With SOAP encoding the content is defined using an encoding scheme which implies a specific mapping to language-specific types.

literal With literal encoding the SOAP content is defined explicitly by some schema (generally XML Schema).

SOAP style and encoding combinations

Three of the four possible SOAP style and encoding combinations are supported by the WSDL importer and the WSDL generator:

- RPC and SOAP encoded (supported for the WSDL importer only).
- RPC and literal.
- Document and literal.

SOAP versions

Two versions of SOAP are available:

- SOAP 1.1
- SOAP 1.2

SOAP 1.1 has some interoperability issues, mainly concerned with the use of SOAP encoding, which are addressed by a separate standard: the WS-I Basic Profile.

Further information

For more information about WSDL 1.1 refer to the World Wide Web Consortium (W3C), and in particular the SOAP 1.1 and SOAP 1.2 documents at:

- http://www.w3.org
- http://www.w3.org/TR/soap

For more information about the WS-I Basic Profile refer to the WS-I, and in particular the WS-I Basic Profile document:

- http://www.ws-i.org/

- http://www.ws-i.org/deliverables

What is WSDL?

WSDL is an XML notation for describing a Web service. A WSDL definition tells a client how to compose a Web service request and describes the interface that is provided by the Web service provider.

WebSphere Message Broker supports WSDL 1.1, as defined in the following document issued by the World Wide Web Consortium (W3C): Web Services Description Language (WSDL) 1.1. WebSphere Message Broker support for WSDL also adheres to the Web Services Interoperability Organization (WS-I) Basic profile 1.1; see Web Services Interoperability Organization (WS-I).

A WSDL definition is divided into separate sections that specify the logical interface and the physical details of a Web service. The physical details include both endpoint information, such as HTTP port number, and binding information, which specifies how the SOAP payload is represented and which transport is used.

Support for WSDL in WebSphere Message Broker includes:
- Import of WSDL to create message definitions in a message set; see Importing from WSDL.
- Generation of WSDL from a message set; see WSDL generation.
- WSDL editor with text and graphical design views.
- Use of WSDL to configure nodes in the SOAP domain; for example, you can drag WSDL onto a node. For more details, see "Using WSDL to configure message flows" on page 746
- Use to WSDL to create a skeleton message flow by dragging WSDL onto the message flow editor canvas. For more details, see "Using WSDL to configure message flows" on page 746.

When you import or generate WSDL, the WSDL is validated against the WS-I Basic Profile. You must fix validation errors before the message set can be deployed. Validation warnings do not prevent deployment, but can indicate potential interoperability problems. The validated WSDL becomes an integral part of the message set.

The WSDL editor supports a graphical design view so that you can navigate from the WSDL to its associated message definitions. The message set contains all the message definitions required by message flows that are working with the Web service described by the WSDL. At development time, the message definitions support ESQL Content Assist and the creation of mappings. At run time, the deployed message set supports schema validation in the SOAP, XMLNSC, and MRM domains. In the SOAP domain, runtime checks are also made against the WSDL itself, and WSDL information is included in the SOAP logical tree.

WSDL validation

The WS-I Validator can be used to check your WSDL definitions against the Basic Profile.

For more information about the WS-I Basic Profile refer to the WS-I, and in particular the WS-I Basic Profile document:
- Web Services Interoperability Organization (WS-I)
- WS-I deliverables index

You can use the WS-I Validator to check your WSDL definitions against the Basic Profile; see “WS-I Basic Profile Version 1.1” on page 809.

You can run the validator in either of the following ways:

- Manually against a specific `.wsdl` resource in the workbench.

  This option enables you to investigate and fix WS-I compliance problems; all validation issues are displayed as task list errors and warnings.
- Automatically, when WSDL is imported or generated.

  WSDL can be imported using the WSDL Quick Start wizard, the WSDL Importer wizard, or the mqsicreatemsgdefsfromwsdl command.

  WSDL can be generated from a message set by using the WSDL Generator wizard.

You can control the behavior of the validator using **Preferences > Web services > Profile Compliance and Validation**. The significant settings are:

- WS-I AP compliance level (WS-I Attachments Profile 1.0)
- WS-I SSBP compliance level (WS-I Simple SOAP Binding Profile 1.0)

You can select one of the following values:

Suggest
: Run the validator with errors treated as unrecoverable, but warnings only notified. This is the default setting.

Require
: Run the validator with errors and warnings treated as unrecoverable.

Ignore Do no run the validator.

The AP selection applies automatically to the SSBP field, therefore Ignore is not explicitly selectable unless the AP selection is set to Ignore.

The following terms refer to the three broad categories of WSDL definition:

- `document-literal` means the combination `style="document"` and `use="literal"`
- `rpc-literal` means the combination `style="rpc"` and `use="literal"`
- `rpc-encoded` means the combination `style="rpc"` and `use="encoded"`

The following are typical validation problems using the preceding terminology:

Your WSDL is rpc-encoded
: WSDL with `use="encoded"` is not WS-I compliant and can lead to operational problems because products of different vendors can make different assumptions about the expected SOAP payload.

  WS-I: (BP2406) The use attribute of a `soapbind:body`, `soapbind:fault`, `soapbind:header`, and `soapbind:headerfault` does not have the value of "literal".

Your WSDL is document-literal, but one or more WSDL part definitions refer to XML Schema types.
: In `document-literal` WSDL, the SOAP body payload is the XML Schema element that is referred to by the appropriate WSDL part.

  If a type is specified instead of an element, the SOAP payload is potentially ambiguous (the payload name is not defined) and interoperability problems are likely.

WS-I: (BP2012) A `document-literal` binding contains `soapbind:body` elements that refer to message part elements that do not have the element attribute.

Your WSDL is rpc-literal or rpc-encoded, but one or more WSDL part definitions refer to XML Schema elements.

In rpc-style WSDL, the SOAP body payload is the WSDL operation name, and its children are the WSDL parts that are specified for that operation.

If an element is specified instead of a type, the SOAP message payload is potentially ambiguous (the payload name might be the WSDL part name or the XML Schema element name), and interoperability problems are likely.

WS-I: (BP2013) An rpc-literal binding contains `soapbind:body` elements that refer to message part elements that do not have the type attribute.

Your WSDL includes SOAP header, headerfault or fault definitions that refer to XML Schema types.

In rpc-style WSDL, the SOAP body is correctly defined through XML Schema types as described above.

SOAP headers and faults, however, do not correspond to an rpc function call in the same way as the body.

In particular, there is no concept of 'parameters' to a header or fault, and a header or fault must always be defined in terms of XML Schema elements to avoid potential ambiguity. Effectively, header and fault definitions in WSDL are always `document-literal`.

WS-I: (BP2113) The `soapbind:header`, `soapbind:headerfault`, or `soapbind:fault` elements refer to `wsd:part` elements that are not defined using only the "element" attribute.

Your WSDL is rpc-literal or rpc-encoded, but no namespace was specified for an operation.

In rpc-style WSDL, the SOAP message payload is the WSDL operation name, qualified by a namespace that is specified as part of the WSDL binding.

If no namespace is specified then the SOAP message payload is potentially ambiguous (the payload name might be in no namespace, or might default to use a different namespace, such as the target namespace of the WSDL definition) and interoperability problems are likely.

WS-I: (BP2020) An rpc-literal binding contains `soapbind:body` elements that either do not have a namespace attribute, or have a namespace attribute value that is not an absolute URI.

Web service interoperability is improved if you implement the following actions:

- Use document-style WSDL whenever possible.
- Use literal encoding, if rpc-style WSDL is necessary.
- Ensure that the WSDL operation definitions are qualified by a valid namespace attribute, if rpc-encoded WSDL must be used.

Using WSDL to configure message flows

You can use WSDL to configure message flows.

Message flows that work with Web services typically use the SOAP nodes. For details about the SOAP nodes, see “WebSphere Message Broker and Web services” on page 735.

The SOAP nodes are configured by using WSDL that was previously imported or generated in a message set, and is displayed under **Deployable WSDL** in the workbench. You can drag the WSDL onto a SOAP node, or specify it by using the WSDL file name property on the node. You must always select a specific WSDL binding.

If you supply a service definition, endpoint properties are set automatically, but you can also set or override these properties manually.

WSDL definitions can optionally be split into multiple files. The typical arrangement is that a top-level service definition file imports a binding file, the binding file imports an interface file, and this interface file imports or includes schema definition files.

A WSDL `portType` (the logical WSDL interface) is not sufficient on its own to configure a SOAP node; a specific binding is required so that the SOAP payload is well-defined at run time.

A binding defines a use, which can be `document` (the default) or `rpc`. If the use is `document`, the SOAP payload is described by an XML Schema element in the WSDL. If the use is `rpc`, the SOAP payload is the WSDL operation name in a specified namespace.

If you want to create your own message flow, configure the nodes as described. However, you can create a new skeleton message flow by dragging a WSDL definition onto a blank **Message Flow Editor** canvas, and selecting a specific WSDL binding. You can also choose the type of flow (service provider or consumer) and the operations to be handled by the flow.

The key nodes and properties in the generated message flow are configured, but you must complete the configuration and add any other nodes you require before deploying the flow. For details about configuring a new Web service by using the wizard, see Configure New Web Service Usage wizard.

Configuring the SOAP nodes

The following nodes are configured explicitly by WSDL:

- “SOAPInput node” on page 1194
- “SOAPRequest node” on page 1204
- “SOAPAsyncRequest node” on page 1172

The following nodes are configured implicitly by WSDL, because they inherit the WSDL configuration of the node with which they are paired:

- “SOAPReply node” on page 1202
- “SOAPAsyncResponse node” on page 1181

A SOAPReply node is always used with a SOAPInput node. For details of Web service scenarios, see “Web services scenarios” on page 827.

A SOAPAsyncResponse node is always used with a SOAPAsyncRequest node, associated by the Unique Identifier property. For SOAP node usage patterns, see "Web services scenarios" on page 827.

What is SOAP MTOM?

SOAP Message Transmission Optimization Mechanism (MTOM) is the use of MIME to optimize the bitstream transmission of SOAP messages that contain significantly large base64Binary elements.

The MTOM message format allows bitstream compression of binary data. Data that would otherwise have to be encoded in the SOAP message is instead transmitted as raw binary data in a separate MIME part. A large chunk of binary data takes up less space than its encoded representation, so MTOM can reduce transmission time, although it can increase processor usage. Candidate elements to be transmitted in this way are defined as base64Binary in the WSDL (XML Schema).

An MTOM message is identified by a Content-Type with a type of `application/xop+xml`.

The SOAP domain handles inbound MTOM messages automatically, and MTOM parts are reincorporated automatically into the SOAP Body.

The use of outbound MTOM messages can be configured on the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes; for details, see "Using SOAP MTOM with the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes."

For details of the external specification published by the World Wide Web Consortium (W3C), see "SOAP MTOM" on page 808.

Using SOAP MTOM with the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes

The use of outbound MTOM messages can be configured on the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes.

The nodes have a property called Allow MTOM, which defines whether MTOM can be used.

An MTOM output message is written if all of the following criteria are met:

- The Allow MTOM property is selected on the **WS Extensions** tab.
- Validation is enabled. The Validate property on the SOAPRequest and SOAPAsyncRequest nodes controls validation of the anticipated response message and not validation of the outgoing request. MTOM output is therefore suppressed unless you set Validate to Content and value on a preceding input node or transformation node.
- No child elements exist below `SOAP.Attachment` in the logical tree. If child elements are present, SOAP with Attachments (SwA) is used.
- Elements exist in the output message that are identified as base64Binary in the associated XML Schema and whose length does not fall below a default threshold size of 1000 bytes.

You can use the local environment setting `MTOMThreshold` to override the MTOM element size threshold. The MTOM element size threshold is set to a default value of 1000 bytes.

The Allow MTOM node property and the `MTOMThreshold` setting can both be overridden in the local environment.

The overrides that apply at a SOAPReply node are:

- `LocalEnvironment.Destination.SOAP.Reply.AllowMTOM`, which can have a value of true or false
- `LocalEnvironment.Destination.SOAP.Reply.MTOMThreshold`, which is an integer value in bytes

The equivalent overrides for a SOAPRequest or SOAPAsyncRequest node are:

- `LocalEnvironment.Destination.SOAP.Request.AllowMTOM`, which can have a value of true or false
- `LocalEnvironment.Destination.SOAP.Request.MTOMThreshold`, which is an integer value in bytes

WS-Addressing

Web Services Addressing (WS-Addressing) is a World Wide Web Consortium (W3C) specification that aids interoperability between Web services by defining a standard way to address Web services and provide addressing information in messages.

Start here to find out how WebSphere Message Broker supports WS-Addressing.

The WS-Addressing specification introduces two primary concepts: endpoint references, and message addressing properties. This topic contains an overview of each concept. For further details, select the following links to access the WS-Addressing specifications:

- W3C WS-Addressing specifications
- W3C submission WS-Addressing specification

Endpoint references (EPRs)

EPRs provide a standard mechanism to encapsulate information about specific endpoints. EPRs can be propagated to other parties and then used to target the Web service endpoint that they represent. The following table summarizes the information model for EPRs.

Abstract Property name	Property type	Multiplicity	Description
[address]	xs:anyURI	1..1	The absolute URI that specifies the address of the endpoint.
[reference parameters]*	xs:any	0..unbounded	Namespace qualified element information items that are required to interact with the endpoint.
[metadata]	xs:any	0..unbounded	Description of the behavior, policies and capabilities of the endpoint.

The following prefix and corresponding namespace is used in the previous table.

Prefix	Namespace
xs	http://www.w3.org/2001/XMLSchema

The following XML fragment illustrates an endpoint reference. This element references the endpoint at the URI http://example.com/fabrikam/acct, has metadata specifying the interface to which the endpoint reference refers, and has application-defined reference parameters of the http://example.com/fabrikam namespace.

```
<wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing"
                       xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
                       xmlns:fabrikam="http://example.com/fabrikam"
                       xmlns:wsdli="http://www.w3.org/2005/08/wsdl-instance"
                       wsdli:wsdlLocation="http://example.com/fabrikam
                       http://example.com/fabrikam/fabrikam.wsdl">
    <wsa:Address>http://example.com/fabrikam/acct</wsa:Address>
    <wsa:Metadata>
        <wsaw:InterfaceName>fabrikam:Inventory</wsaw:InterfaceName>
    </wsa:Metadata>
    <wsa:ReferenceParameters>
        <fabrikam:CustomerKey>123456789</fabrikam:CustomerKey>
        <fabrikam:ShoppingCart>ABCDEFG</fabrikam:ShoppingCart>
    </wsa:ReferenceParameters>
</wsa:EndpointReference>
```

Message addressing properties (MAPs)

MAPs are a set of well defined WS-Addressing properties that can be represented as elements in SOAP headers. MAPs can provide either a standard way of conveying information, such as the endpoint to which message replies should be directed, or information about the relationship that the message has with other messages. The MAPs that are defined by the WS-Addressing specification are summarized in the following table.

Abstract WS-Addressing MAP name	MAP content type	Multiplicity	Description
[action]	xs:anyURI	1..1	An absolute URI that uniquely identifies the semantics of the message. This proprety corresponds to the [address] property of the endpoint reference to which the message is addressed. This value is required.
[destination]	xs:anyURI	1..1	The absolute URI that specifies the address of the intended receiver of this message. This value is optional because, if not present, it defaults to the anonymous URI that is defined in the specification, indicating that the address is defined by the underpinning protocol.
[reference parameters]*	xs:any	0..unbounded	Correspond to the [reference parameters] property of the endpoint reference to which the message is addressed. This value is optional.
[source endpoint]	EndpointReference	0..1	A reference to the endpoint from which the message originated. This value is optional.
[reply endpoint]	EndpointReference	0..1	An endpoint reference for the intended receiver of replies to this message. This value is optional.
[fault endpoint]	EndpointReference	0..1	An endpoint reference for the intended receiver of faults relating to this message. This value is optional.

Abstract WS-Addressing MAP name	MAP content type	Multiplicity	Description
[relationship]*	xs:anyURI plus optional attribute of type xs:anyURI	0..unbounded	A pair of values that indicate how this message relates to another message. The content of this element conveys the [message id] of the related message. An optional attribute conveys the relationship type. This value is optional.
[message id]	xs:anyURI		An absolute URI that uniquely identifies the message. This value is optional.

The abstract names in the previous tables are used to refer to the MAPs throughout this documentation.

The following example of a SOAP message contains WS-Addressing MAPs:

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
            xmlns:wsa="http://www.w3.org/2005/08/addressing"
            xmlns:fabrikam="http://example.com/fabrikam">
    <S:Header>
        ...
     <wsa:To>http://example.com/fabrikam/acct</wsa:To>
              <wsa:ReplyTo>
        <wsa:Address> http://example.com/fabrikam/acct</wsa:address>
              </wsa:ReplyTo>
            <wsa:Action>...</wsa:Action>
              <fabrikam:CustomerKey wsa:IsReferenceParameter='true'>123456789
              </fabrikam:CustomerKey>
             <fabrikam:ShoppingCart wsa:IsReferenceParameter='true'>ABCDEFG
             </fabrikam:ShoppingCart>
              ...
    </S:Header>
        <S:Body>
          ...
        </S:Body>
</S:Envelope>
```

How to use WS-Addressing

An overview of how you use WS-Addressing with WebSphere Message Broker.

Sending a message to an endpoint reference (EPR)

When sending a message to an endpoint reference, the following processes take place:

- The [destination] Message Addressing Property (MAP) is filled in from the [address] property in the EPR.
- [reference parameters] are copied to top level SOAP headers from the [reference parameters] property of the EPR.
- The [action] property is required, but is not populated from the EPR.

  In this context, action is an absolute Internationalized Resource Identifier (IRI) that uniquely identifies the semantics implied by this message, and it must be the same as the HTTP SOAPAction if a non-empty SOAPAction is specified.
- The [message id] property must be specified if this message is part of a request-response Message Exchange Pattern (MEP); the message id is generated by default.

When replying with a non-fault message, the following processes take place:

- The EPR to reply to is selected from the [reply endpoint] MAP.

- If this property contains the special address `none`, no reply is sent.
- If this property contains the special address `anonymous`, the reply is sent on the return channel of the transport on which the request was received. This is the default value in the absence of any other supplied EPR.
- Otherwise, the reply is sent to the [`address`] property in the Reply EPR,

- The [`message id`] property of the inbound message request is placed into the [`relationship`] property of the response, along with the predefined relationship of the reply part of the Universal Resource Identifier (URI) - which indicates that this message is a reply.

  For further information on the URI see Web Services Addressing URI Specification.
- A new [`message id`] property is specified for the reply, and this is generated by default.

WS-Addressing with the SOAPInput node

Various options are available when you use WS-Addressing with the SOAPInput node.

The SOAPInput node has a property for processing WS-Addressing information present in the incoming message called `Use WS-Addressing`.

If you select this property, the WS-Addressing information is processed and the process itself is called `engaging WS-Addressing`. The default is that WS-Addressing is not engaged.

You can also specify this property in the WSDL, and this property is configurable from the WSDL, automatically by the workbench, when the WSDL is dropped onto the node. The behavior of the node when WS-Addressing is engaged or not engaged is as follows:

Addressing not engaged

No WS-Addressing processing is performed. If a message is received that contains any WS-Addressing headers they are ignored, and no WS-Addressing processing of any kind is performed, unless they are marked as `MustUnderstand`.

The inbound WS-Addressing headers in this case are visible in the message when it leaves the SOAPInput node under the Header folder of the SOAP parser in the message tree.

A fault is returned to the client if WS-Addressing headers exist in the incoming message, and they meet both of the following criteria:

- Marked as `MustUnderstand`
- Targeted at the role the SOAPInput node is operating in

Engaging WS-Addressing is how you instruct the node to 'understand' the WS-Addressing headers. In this case the WS-Addressing headers remain in the SOAP Header section of the SOAP parser, and no other SOAP node acts upon them. In all cases, they are treated as a SOAP header with no special meaning assigned to the WS-Addressing headers.

Addressing engaged:

WS-Addressing processing is performed as stated in the WS-Addressing specification. This processing means that messages that contain either submission addressing headers or final addressing headers are accepted.

A fault is returned if both submission addressing headers and final headers are present, and either of the following conditions is met:

- Neither is marked with a role.
- They are both marked with same role and the SOAPInput node is acting in that role.

Assuming that the WS-Addressing headers are valid and the Place WS-Addressing Headers into LocalEnvironment check box is selected on the SOAPInput node, all headers (including detectable inbound reference parameters) are removed from the inbound message tree and are placed into the local environment tree under the `SOAP.Input.WSA` folder. Moving the WS-Addressing headers to the local environment indicates that they have been processed by the broker. The headers are removed from the message tree because they have been processed on input; otherwise they would not be valid if the message tree was sent out without further changes. They are stored in the local environment to allow you to inspect them.

Only reference parameters from the final specification are detectable because they have an attribute called `IsReferenceParameter` that allows them to be detected. Submission reference parameter headers do not have this attribute, therefore they are not detectable, and they are not moved into the local environment tree from the message tree.

You can change WS-Addressing reply headers before the SOAPReply node is reached. For more information about changing WS-Addressing information in the local environment, see "WS-Addressing information in the local environment" on page 756.

WS-Addressing with the SOAPReply node

Various options are available when you use WS-Addressing with the SOAPReply node.

The SOAPReply node uses WS-Addressing if WS-Addressing is engaged on the SOAPInput node that is referenced by the reply identifier of the message entering the reply node.

The SOAPReply node uses addressing information in the `Destination.SOAP.Reply.WSA` folder of the local environment to determine where to send the reply and with what Message Addressing Properties (MAPs).

If the `Destination.SOAP.Reply.WSA` does not exist, or is completely empty when inspected by the SOAPReply node, the node uses the default addressing headers that were part of the incoming message. Therefore, you do not have to propagate the local environment in the default case, and addressing still works as expected.

In the case where folders exist beneath the `Reply.WSA` folder, these folders are used to update the output message. Therefore, you can change, add, or remove parts of the default reply information generated by the input node, because any changes that you made to the tree are reflected in the outgoing message by the SOAPReply node. For details about WS-Addressing information in the local environment, see "WS-Addressing information in the local environment" on page 756.

If WS-Addressing is not engaged, this node does not perform any WS-Addressing processing.

WS-Addressing with the SOAPRequest node

Various options are available when you use WS-Addressing with the SOAPRequest node.

The SOAPRequest node has a property called `Use WS-Addressing`, for processing WS-Addressing information that is present in the incoming message.

If you select this property, the WS-Addressing information is processed and the process itself is called `engaging WS-Addressing`. The default is that WS-Addressing is not engaged.

You can also specify this property in the WSDL and this is configurable from the WSDL, automatically by the workbench, when the WSDL is dragged onto the node. The behavior of the node when WS-Addressing is engaged or not is as follows:

Addressing not engaged
The node does not add any WS-Addressing headers to the outgoing message, and does not process any WS-Addressing headers that might be present in the response message that is received by the node.

Addressing engaged:
The node first looks at the `Destination.SOAP.Request.WSA` folder in the local environment. If this folder is empty, the node automatically generates all required WS-Addressing Message Addressing Properties (MAPs) in the outgoing message, by using the following default values:

- `Action`, from the WSDL configuration file. If this is not explicitly specified, this defaults to the value that is defined in the WSDL Binding specification.
- `To`, from the `Web Service URL` node property.
- `ReplyTo`, by using the special `Anonymous` address (assuming that the Operation being used is not a one-way message exchange program, in which case a `ReplyTo` by using the special `None` address is specified).
- `MessageID`, a unique UUID is used.

If the `Destination.SOAP.Request.WSA` folder in the LocalEnvironment is not empty, any user supplied MAPs override the default ones that were listed previously, on a property by property basis.

After the response to the request is received and if the Place WS-Addressing Headers into LocalEnvironment check box is selected on the SOAPRequest node, the SOAPRequest node removes all WS-Addressing headers from the response message and places them in the `SOAP.Response.WSA` folder. This folder allows you to query the headers in a similar manner to the way the SOAPInput node deals with the Input WS-Addressing headers.

WS-Addressing with the SOAPAsyncRequest and SOAPAsyncResponse nodes

The remote Web service must understand WS-Addressing to be able to work with SOAPAsyncRequest and SOAPAsyncResponse nodes.

The SOAPAsyncRequest and SOAPAsyncResponse nodes require WS-Addressing; therefore, the remote Web service must understand WS-Addressing to process the WS-Addressing headers that are sent from the SOAPAsyncRequest node, and to

allow the response to be sent back to the corresponding SOAPAsyncResponse node, which is specified in the address property of the ReplyTo Message Addressing Property (MAP).

SOAPAsyncRequest node

The SOAPAsyncRequest node has a property called Use WS-Addressing that is read-only and has a default value of true, indicating that WS-Addressing is mandatory for this node. This property has the effect of permanently engaging WS-Addressing for this node and cannot be changed by the node, or by the WSDL that is used to configure this node.

The node first looks at the Destination.SOAP.Request.WSA folder in the local environment. If this folder is empty, the node automatically generates all required WS-Addressing MAPs in the outgoing message, using the following default values:

- Action, from the WSDL configuration file. If this value is not specified explicitly, the default value is defined by the WSDL Binding specification.
- To, from the Web Service URL node property.
- ReplyTo, the address of the corresponding SOAPAsyncResponse node.
- MessageID, a unique UUID is used.

If the Destination.SOAP.Request.WSA folder in the local environment is not empty, any user-supplied MAPs override the default ones listed previously on a property by property basis.

However, because of the nature of the SOAP asynchronous node pair, you cannot specify the address property of the ReplyTo Message Exchange Program (MEP), and this property is ignored if specified.

When the main MAPs are generated, the node looks in several places to obtain various pieces of context information to send in a <wmb:context> element under the ReferenceParameters section of the ReplyTo endpoint reference. If these locations exist and are not empty, the following additional information is added to the <wmb:context>:

- Destination.SOAP.Request.UserContext

  This information is added under a subfolder called UserContext.
- Destination.SOAP.Reply.ReplyIdentifier

  This information is added under a subfolder called ReplyID.

Use the user context to specify an arbitrary amount of data that will be sent with the message from the SOAPAsyncRequest node to the SOAPAsyncResponse node. By using the user context, you can pass state from one node to the other. Ensure that the amount of data that you send is small because this data is placed in the message.

Use the reply identifier to automatically correlate a SOAPInput node in the flow that contains the SOAPAsyncRequest node, with a SOAPReply node in the flow that contains the SOAPAsyncResponse node.

SOAPAsyncResponse node

After the response to the request is received, the SOAPAsyncResponse node removes all WS-Addressing headers from the response message and places them in the SOAP.Response.WSA folder so that you can query the headers.

If the response message contains a user context that was specified by the SOAPAsyncRequest node, the user context is placed in the `SOAP.Response.UserContext` folder in the local environment.

If the response message contains a reply identifier that was specified by the SOAPAsyncRequest node, the reply identifier is placed in the `Destination.SOAP.Reply.ReplyIdentifier` folder in the local environment.

WS-Addressing information in the local environment

WS-Addressing header information can be placed in the local environment tree where it is visible to a message flow. WS-Addressing header information is only processed by the SOAP nodes.

Inbound messages

Inbound information is placed in the local environment by the SOAP node only if addressing is engaged on the node and you select the Place WS-Addressing Headers into LocalEnvironment property on the SOAPInput, SOAPAsyncResponse, or SOAPRequest nodes.

The following table describes the node specific WS-Addressing information in the local environment tree.

Node	Populates local environment property
SOAPInput	`LocalEnvironment.SOAP.Input.WSA.`*`type`*
SOAPAsyncResponse	`LocalEnvironment.SOAP.Response.WSA.`*`type`*
SOAPRequest	`LocalEnvironment.SOAP.Request.WSA.`*`type`*

Where *type* is the structure of the subsection of the local environment WS-Addressing XML schema. For details about how *type* maps to the WS-Addressing properties defined by the WS-Addressing specification, see the "Local environment property type" on page 757 section of this topic.

The local environment information for inbound messages is for your information only. If you engage addressing on the node, and select the Place WS-Addressing Headers into LocalEnvironment property on the node, WS-Addressing information is available for you to look at and use in your flow. The WS-Addressing properties are placed in the local environment after processing by the node. Note that the WS-Addressing folder and all its children are owned by an XMLNSC parser, therefore you can copy elements directly into any other tree that is owned by an XMLNSC parser. However, be aware that if you copy this folder (or any of its children) to a tree that is not owned by an XMLNSC parser, information in the tree is discarded unless you create an XMLNSC parser in the target tree first. This behavior can occur if you, for example, copy from the `InputLocalEnvironment` tree to the `OutputLocalEnvironment` tree.

Outbound messages

You can place outbound WS-Addressing header information in the local environment; however, this practice is necessary only to override the defaults that are generated by the node automatically . Outbound addressing headers are created only if WS-Addressing is enabled on the node.

The following table describes the node specific WS-Addressing information in the local environment tree that can be used to override the defaults for outbound messages.

Node	Populates local environment property
SOAPReply	`LocalEnvironment.Destination.SOAP.Reply.WSA.`*type*
SOAPRequest	`LocalEnvironment.Destination.SOAP.Request.WSA.`*type*
SOAPAsyncRequest	`LocalEnvironment.Destination.SOAP.Request.WSA.`*type*

Where *type* is the structure of the subsection of the local environment WS-Addressing XML schema. For details about how the *type* maps to the WS-Addressing properties defined by the WS-Addressing specification, see the "Local environment property type" section of this topic.

You can modify local environment information for outbound messages. The SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes generate default local environment settings that you can override. One exception to this table is that any attempt to override the WS-Addressing `ReplyTo` address on the SOAPAsyncRequest node is ignored.

For example, to set WS-Addressing information in the local environment for the SOAPRequest node:

```
 SET OutputRoot = InputRoot;
 SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.To.Address = 'jms:jndi:INPUTQ';
 SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.ReplyTo.Address = 'jms:jndi:RESPONSEQ?jndiConnectionFactoryName=
jndiInitialContextFactory=com.sun.jndi.fscontext.RefFSContextFactory&
jndiURL=file://C:/SOAPJNDIBindings';
 SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.From.Address = 'jms:jndi:INPUTQ';
 SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.FaultTo.Address = 'jms:jndi:RESPONSEQ?jndiConnectionFactoryName=
jndiInitialContextFactory=com.sun.jndi.fscontext.RefFSContextFactory&
jndiURL=file://C:/SOAPJNDIBindings';
 SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.Action = 'http://WMB_BankImport/NewOperation';
 SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.MessageID = 'test:my:msg:ID:1234578';
 SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.IncludeOptionalHeaders = false;
```

Local environment property *type*

The local environment property *type* in the preceding tables corresponds to the WS-Addressing part of the local environment XML schema. The following table shows the corresponding message addressing properties (MAPs) of the WS-Addressing local environment schema for all nodes.

Element	Corresponds to abstract WS-Addressing MAP name
To	[destination endpoint]
From	[source endpoint]
ReplyTo	[reply endpoint]
FaultTo	[fault endpoint]
Action	[action]
MessageId	[message id]
RelatesTo	[relationship]
ReferenceParameters	[reference parameters]

Element	Corresponds to abstract WS-Addressing MAP name
Version	This element does not correspond to a MAP, but it is used to identify the version of WS-Addressing. The two main versions of WS-Addressing are Submission and Final. The default version that is used by all nodes is Final. Therefore, for outbound messages, set this element only if you want the version to be Submission. For incoming messages, this element is populated automatically with the version of the WS-Addressing headers that the inbound message used.

For more details about the message addressing properties defined by the WS-Addressing specification, see "WS-Addressing" on page 749.

For outbound WS-Addressing, you can set an additional local environment property.

Element	Description
AddMustUnderstandAttribute	This element places the SOAP `mustUnderstand` attribute on each WS-Addressing header before the message is sent.

Example use of WS-Addressing information in the local environment

This example shows the setting of reference parameters in the local environment along with the corresponding messages as they appear on the wire.

In this example the Web service exposes a simple ping operation.

ESQL to add reference parameters

The following ESQL example shows how to specify addressing headers in the local environment.

```
DECLARE Example_ns NAMESPACE 'http://ibm.namespace';

SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.ReplyTo.ReferenceParameters.Parameter1 = 'Message Broker';
SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.ReplyTo.ReferenceParameters.Example_ns:Parameter2.
 (SOAP.NamespaceDecl)xmlns:Example_ns = 'http://ibm.namespace';
SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.ReplyTo.ReferenceParameters.Example_ns:Parameter2 = 'Ping';

SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.FaultTo.ReferenceParameters.Parameter1 = 'Ping';
SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.FaultTo.ReferenceParameters.Example_ns:Parameter2.
 (SOAP.NamespaceDecl)xmlns:Example_ns = 'http://ibm.namespace';
SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.FaultTo.ReferenceParameters.gns:Parameter2 = 'FAULT';
```

Request message

The following example is of an outgoing SOAP envelope in a message from a SOAPRequest node with `ReplyTo` and `FaultTo` reference parameters generated after using the above ESQL. It also shows the other message addressing properties (MAPs) that are not set in the local environment, but are generated automatically by the node as a result of engaging WS-Addressing.

```
<NS1:Envelope xmlns:NS1="http://www.w3.org/2003/05/soap-envelope" xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <NS1:Header>
    <wsa:To>http://localhost:7801/Service</wsa:To>
    <wsa:ReplyTo>
      <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
      <wsa:ReferenceParameters>
        <Example_ns:Parameter2 xmlns:Example_ns="http://ibm.namespace">Ping</Example_ns:Parameter2>
        <Parameter1>Message Broker</Parameter1>
      </wsa:ReferenceParameters>
```

```
    </wsa:ReplyTo>
    <wsa:FaultTo>
      <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
      <wsa:ReferenceParameters>
        <Example_ns:Parameter2 xmlns:Example_ns="http://ibm.namespace">FAULT</Example_ns:Parameter2>
        <Parameter1>Ping</Parameter1>
      </wsa:ReferenceParameters>
    </wsa:FaultTo>
    <wsa:MessageID>urn:uuid:020C911C16EB130A8F1204119836321</wsa:MessageID>
    <wsa:Action>http://ibm.com/Service/Ping</wsa:Action>
  </NS1:Header>
  <NS1:Body>
    <NS2:Ping xmlns:NS2="http://ibm.com"></NS2:Ping>
  </NS1:Body>
</NS1:Envelope>
```

In the above example, reference parameters are set for the `ReplyTo` and `FaultTo` endpoint references (EPRs). If this message is sent to a SOAPInput node with WS-Addressing engaged, these `ReferenceParameters` are placed in the local environment of the flow that contains the SOAPInput node for use by the flow if the Place WS-Addressing Headers into LocalEnvironment property is selected. This option changes only what is placed in the local environment; it does not change the contents of the response message.

Response message

The following SOAP envelope is a response to the outgoing preceding message, as sent by a SOAPReply node. This example shows the MAP processing that happens automatically by the SOAPReply node. In this example, the `FaultTo` reference parameters are not present because the reply is not a SOAP fault. This response also shows where the reference parameters that belonged to the `ReplyTo` EPR appear in the response message.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wsa="http://www.w3.org/2005/08/
 addressing">
  <soapenv:Header>
    <wsa:To>http://www.w3.org/2005/08/addressing/anonymous</wsa:To>
    <Example_ns:Parameter2 wsa:IsReferenceParameter="true" xmlns:Example_ns="http://ibm.namespace">Ping</Example_ns:
 Parameter2>
    <Parameter1 wsa:IsReferenceParameter="true">Message Broker</Parameter1>
    <wsa:ReplyTo>
      <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
    </wsa:ReplyTo>
    <wsa:Action>http://ibm.com/Service/PingResponse</wsa:Action>
    <wsa:RelatesTo RelationshipType="http://www.w3.org/2005/08/addressing/reply">urn:uuid:020C911C16EB130A8F120411983632
 </wsa:RelatesTo>
  </soapenv:Header>
  <soapenv:Body>
    <NS1:PingResponse xmlns:NS1="http://ibm.com">
      <NS1:PingResult>Ping</NS1:PingResult>
    </NS1:PingResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

WS-Security

Web Services Security (WS-Security) describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. WS-Security mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

WS-Security is a message-level standard that is based on securing SOAP messages through XML digital signature, confidentiality through XML encryption, and credential propagation through security tokens. The Web services security specification defines the facilities for protecting the integrity and confidentiality of a message and provides mechanisms for associating security-related claims with the message.

WS-Security provides a general-purpose mechanism for associating security tokens with messages. No specific type of security token is required by WS-Security. It is designed to be extensible, for example, to support multiple security token formats.

WS-Security also describes how to encode binary security tokens and attach them to SOAP messages. Specifically, the WS-Security profile specifications describe how to encode the following tokens:

- Username tokens
- X.509 certificates

With WS-Security, the domain of these mechanisms can be extended by carrying authentication information in Web services requests. WS-Security also includes extensibility mechanisms that can be used to further describe the credentials that are included with a message. WS-Security is a building block that can be used in conjunction with other Web service protocols to address a wide variety of application security requirements.

There are numerous advantages to using WS-Security.

- Different parts of a message can be secured in a variety of ways. For example, you can use integrity on the security token (user ID and password) and confidentiality on the SOAP message body.
- Intermediaries can be used and end-to-end message-level security can be provided through any number of intermediaries.
- WS-Security works across multiple transports and is independent of the underlying transport protocol.
- Authentication of both individual users and multiple party identities is possible.

Traditional Web security mechanisms, such as HTTPS, might be insufficient to manage the security requirements of all Web service scenarios. For example, when an application sends a SOAP message using HTTPS, the message is secured only for the HTTPS connection, meaning during the transport of the message between the service requester (the client) and the service. However, the application might require that the message data be secured beyond the HTTPS connection, or even beyond the transport layer. By securing Web services at the message level, message-level security is capable of meeting these expanded requirements.

Message-level security, or securing Web services at the message level, addresses the same security requirements as for traditional Web security. These security requirements include: identity, authentication, authorization, integrity, confidentiality, nonrepudiation, and basic message exchange. Both traditional Web and message-level security share many of the same mechanisms for handling security, including digital certificates, encryption, and digital signatures.

With message-level security, the SOAP message itself either contains the information needed to secure the message or it contains information about where to get that information to handle security needs. The SOAP message also contains information relevant to the protocols and procedures for processing the specified

message-level security. However, message-level security is not tied to any particular transport mechanism. Because the security information is part of the message, it is independent of a transport protocol, such as HTTPS.

The client adds to the SOAP message header security information that applies to that particular message. When the message is received, the Web service endpoint, using the security information in the header, verifies the secured message and validates it against the policy. For example, the service endpoint might verify the message signature and check that the message has not been tampered with. It is possible to add signature and encryption information to the SOAP message headers, as well as other information such as security tokens for identity (for example, an X.509 certificate) that are bound to the SOAP message content.

Without message-level security, the SOAP message is sent in clear text, and personal information such as a user ID or an account number is not protected. Without applying message-level security, there is only a SOAP body under the SOAP envelope in the SOAP message. By applying features from the WS-Security specification, the SOAP security header is inserted under the SOAP envelope in the SOAP message when the SOAP body is signed and encrypted.

To keep the integrity or confidentiality of the message, digital signatures and encryption are typically applied.

- Confidentiality specifies the confidentiality constraints that are applied to generated messages. This includes specifying which message parts within the generated message must be encrypted, and the message parts to attach encrypted Nonce and time stamp elements to.
- Integrity is provided by applying a digital signature to a SOAP message. Confidentiality is applied by SOAP message encryption.

You can add an authentication mechanism by inserting various types of security tokens, such as the Username token (element). When the Username token is received by the Web service server, the user name and password are extracted and verified. Only when the user name and password combination is valid, will the message be accepted and processed at the server. Using the Username token is just one of the ways of implementing authentication. This mechanism is also known as basic authentication.

The OASIS Web Services Security Specification provides a set of mechanisms to help developers of Web Services secure SOAP message exchanges. For details of the OASIS Web Services Security Specification, see OASIS Standard for WS-Security Specification.

WS-Security mechanisms

The WS-Security specification provides three mechanisms for securing Web services at the message level: authentication, integrity, and confidentiality.

Authentication

This mechanism uses a security token to validate the user and determine whether a client is valid in a particular context. A client can be a user, computer, or application. Without authentication, an attacker can use spoofing techniques to send a modified SOAP message to the service provider.

In authentication, a security token is inserted in the request message. Depending on the type of security token that is being used, the security token can also be inserted in the response message. The following types of security token are supported for authentication:
- Username tokens
- X.509 certificates

Username tokens are used to validate user names and passwords. When a Web service server receives a username token, the user name and password are extracted and passed to a user registry for verification. If the user name and password combination is valid, the result is returned to the server and the message is accepted and processed. When used in authentication, username tokens are typically passed only in the request message, not the response message.

X.509 tokens are validated by using a certificate path.

All types of token must be protected. For this reason, if you send them over an untrusted network, take one of the following precautions:
- Use HTTPS
- Configure the policy set to protect the appropriate elements in the SOAP header

Integrity

This mechanism uses message signing to ensure that information is not changed, altered, or lost accidentally. When integrity is implemented, an XML digital signature is generated on the contents of a SOAP message. If unauthorized changes are made to the message data, the signature is not validated. Without integrity, an attacker can use tampering techniques to intercept a SOAP message between the Web service client and server, and modify it.

Confidentiality

This mechanism uses message encryption to ensure that no party or process can access or disclose the information in the message. When a SOAP message is encrypted, only a service that knows the appropriate key can decrypt and read the message.

Implementing WS-Security

Configure authentication, XML encryption, XML signature, and message expiration by using the Policy Sets and Policy Set Bindings editor.

You use the Policy Sets and Policy Set Bindings editor in the workbench to configure the following aspects of WS-Security:

"Authentication"

"Confidentiality" on page 763

"Integrity" on page 764

"Expiration" on page 764

Authentication

The following tokens are supported:
- Username
- X.509

Configuring authentication with username tokens:

1. In the workbench, switch to the Broker Administration perspective.
2. Create a policy set and add UserName authentication tokens to it; see Policy Sets and Policy Set Bindings editor: Authentication tokens panel.
3. Further configure any X.509 authentication tokens defined in the associated policy set; see Policy Sets and Policy Set Bindings editor: Authentication and Protection Tokens panel.
4. Configure a security profile; see "Message flow security and security profiles" on page 774.
5. Associate the policy set with a message flow or node; see "Associating policy sets and bindings with message flows and nodes" on page 772.

Configuring authentication with X.509 tokens:

1. If you are using the broker's truststore to hold the trusted certificate, you must configure it; see "Viewing and setting keystore and truststore runtime properties at broker level" on page 769 or "Viewing and setting keystore and truststore runtime properties at execution group level" on page 771 depending on where you want to set keystore and truststore runtime properties.
2. In the workbench, switch to the Broker Administration perspective.
3. Create a policy set and add UserName and X.509 authentication tokens to it; see Policy Sets and Policy Set Bindings editor: Authentication tokens panel.
4. Configure the certificate mode for either broker truststore or an external security provider; see Policy Sets and Policy Set Bindings editor: Authentication and Protection Tokens panel.
5. If you are using an external security provider, configure a security profile; see "Message flow security and security profiles" on page 774.
6. Associate the policy set with a message flow or node; see "Associating policy sets and bindings with message flows and nodes" on page 772

Confidentiality

Confidentiality is provided by XML encryption, and requires X.509 tokens.

Configuring XML encryption with X.509 tokens:

1. If you are using the broker's truststore to hold the trusted certificate, you must configure it; see "Viewing and setting keystore and truststore runtime properties at broker level" on page 769 or "Viewing and setting keystore and truststore runtime properties at execution group level" on page 771, depending on where you want to set keystore and truststore runtime properties.
2. In the workbench, switch to the Broker Administration perspective.
3. Create a policy set, enable XML encryption, create encryption tokens, and select the encryption algorithms that you will use; see Policy Sets and Policy Set Bindings editor: Message Level Protection panel.
4. Define which parts of a message are to be encrypted; see Policy Sets and Policy Set Bindings editor: Message Part Protection panel.
5. Further configure message part encryption; see Policy Sets and Policy Set Bindings editor: Message Part Policies panel.
6. Further configure the keystore and truststore; see Policy Sets and Policy Set Bindings editor: Key Information panel.
7. Associate the policy set with a message flow or node; see "Associating policy sets and bindings with message flows and nodes" on page 772.

Integrity

Integrity is provided by XML signature, and requires X.509 tokens.

Configuring XML signature with X.509 tokens:

1. If you are using the broker's truststore to hold the trusted certificate, you must configure it; see "Viewing and setting keystore and truststore runtime properties at broker level" on page 769 or "Viewing and setting keystore and truststore runtime properties at execution group level" on page 771 depending on where you want to set keystore and truststore runtime properties.
2. In the workbench, switch to the Broker Administration perspective.
3. In the Properties window, select the Security tab, and click **Policy Sets**.
4. Create a policy set, enable XML signature, and create signature tokens; see Policy Sets and Policy Set Bindings editor: Message Level Protection panel.
5. Define which parts of a message are to be signed; see Policy Sets and Policy Set Bindings editor: Message Part Protection panel.
6. Further configure message part signature; see Policy Sets and Policy Set Bindings editor: Message Part Policies panel.
7. Further configure the keystore and truststore; see Policy Sets and Policy Set Bindings editor: Key Information panel.
8. Associate the policy set with a message flow or node; see "Associating policy sets and bindings with message flows and nodes" on page 772.

Expiration

To configure message expiration, see Policy Sets and Policy Set Bindings editor: Message Expiration panel.

Policy sets

Policy sets and bindings define and configure your WS-Security requirements, supported by WebSphere Message Broker, for the SOAPInput, SOAPReply, SOAPRequest, SOAPAsyncRequest, and SOAPAsyncResponse nodes.

A *policy set* is a container for the WS-Security policy type.

A policy set *binding* is associated with a policy set and contains information that is specific to the environment and platform, such as information about keys.

Use policy sets and bindings to define the following items for both request and response SOAP messages:

- Authentication for both Username tokens and X.509 certificates
- Asymmetric encryption (confidentiality)
- Asymmetric signature (integrity) (requires the broker keystore and truststore)

Either the whole SOAP message body, or specific parts of the SOAP message header and body can be encrypted and signed.

You administer policy sets and bindings from the toolkit, which can add, delete, display and edit policy sets and bindings. Any changes to policy sets or bindings in the toolkit are saved directly to the associated broker. You must stop and then restart the message flow for the new configuration information to take effect.

You can also export and import policy sets and bindings from a broker.

- "Exporting a policy set and policy binding" on page 773
- "Importing a policy set and policy set binding" on page 774

Policy sets and their associated bindings must be saved and restored together.

Policy sets are associated with a message flow, a node or both in the Broker Archive editor. For convenience, you can specify settings for *provider* and *consumer* at the message flow level. The provider setting applies to all SOAPInput and SOAPReply nodes in the message flow. The consumer setting applies to all SOAPRequest, SOAPAsyncRequest, and SOAPAsyncResponse nodes. Individual policy set and binding assignments can be applied at the node level in the Broker Archive editor, and these take precedence over the flow-level provider and consumer settings. The default setting is *none*, meaning that no policy set and bindings are to be used.

Several nodes in the same message flow can refer to the same policy set and bindings. It is the responsibility of the administrator to ensure that the required policy sets are available to the broker at run time. An error is reported if the broker cannot find the associated policy set or bindings.

The rest of this topic describes some of the terms that you will meet when configuring policy sets and bindings.

Default policy set and bindings

When a broker is created, default policy set and bindings are created called WSS10Default. This default contains a limited security policy which specifies that a Username token is present in request messages (inbound) to SOAPInput nodes in the associated message flow. The default policy set binding refers to the default policy set. They are not editable.

Consumer and provider nodes

Nodes are either consumers or providers.

Consumer nodes

SOAPRequest
SOAPAsyncRequest
SOAPAsyncResponse

Provider nodes

SOAPInput
SOAPReply

Request and response

Request and response is a message exchange pattern (MEP). It describes a client that sends a SOAP Request message to a Web services server, which in turn sends a Response SOAP message back to the client. The Request message is always the SOAP message from the client to the server, and the Response message is always the SOAP message reply from server to the client. The following table describes this pattern in relation to the WebSphere Message Broker SOAP nodes:

Node	Broker viewpoint	Request	Response
SOAPInput	SOAP message inbound	Inbound message	Not applicable
SOAPReply	SOAP message outbound	Not applicable	Outbound message
SOAPRequest	SOAP message outbound followed by a SOAP message inbound	Outbound message	Inbound message
SOAPAsyncRequest	SOAP message outbound	Outbound message	Not applicable
SOAPAsyncResponse	SOAP message inbound	Not applicable	Inbound message

Initiator and recipient

Initiator and recipient are roles defined in the exchange of SOAP messages.

Initiator
The role that sends the initial message in a message exchange.

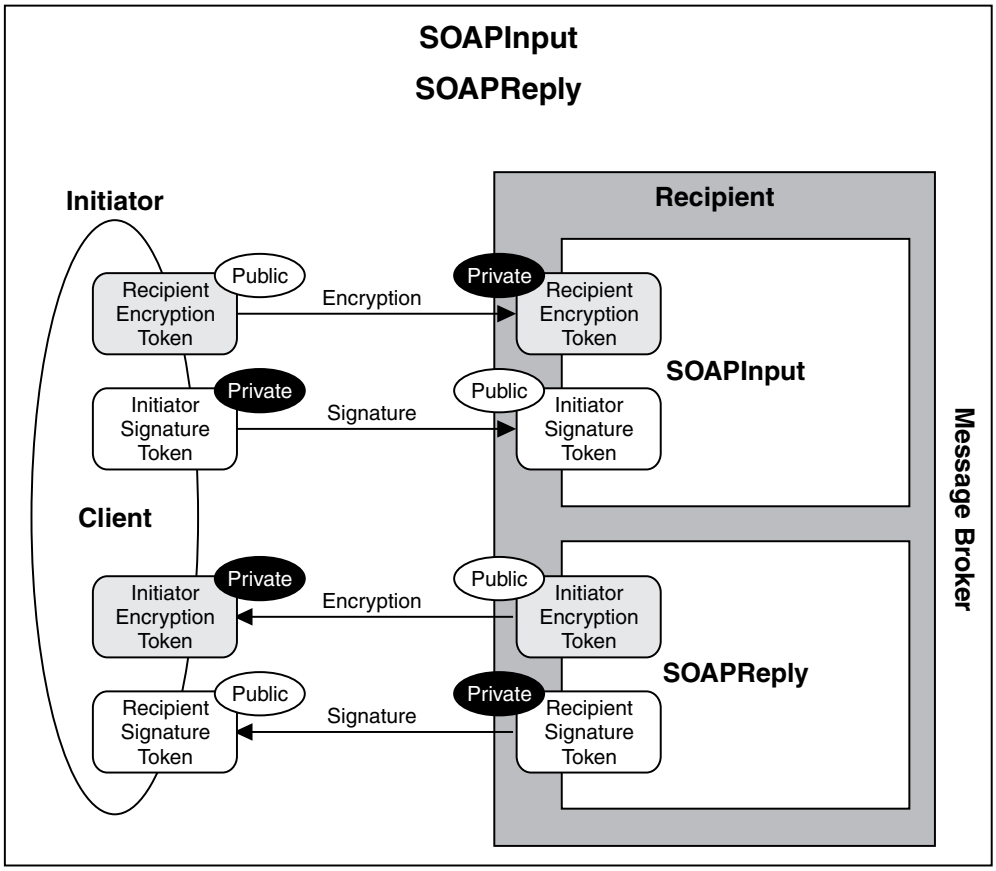
Recipient
The targeted role to process the initial message in a message exchange.

The following table describes these roles in relation to the Message Broker SOAP nodes:

Node	Broker viewpoint	Initiator	Recipient
SOAPInput	SOAP message inbound	External client sending SOAP message to the broker.	SOAPInput node
SOAPReply	SOAP message outbound	External client that sent the original SOAP message to the broker.	SOAPReply node
SOAPRequest (outbound)	SOAP message outbound followed by a SOAP message inbound	SOAPRequest node	External provider receiving the SOAP message
SOAPRequest (inbound)	SOAP message outbound followed by a SOAP message inbound	SOAPRequest node	External provider receiving the SOAP message
SOAPAsyncRequest	SOAP message outbound	SOAPAsyncRequest node	External provider receiving the SOAP message
SOAPAsyncResponse	SOAP message inbound	SOAPAsyncRequest node	External provider receiving the SOAP message

SOAPInput and SOAPReply nodes

In this diagram, the broker acts as recipient. A SOAPInput node receives a message from a client (initiator). A SOAPReply node replies. Inbound and outbound messages are signed and encrypted.



In the request:

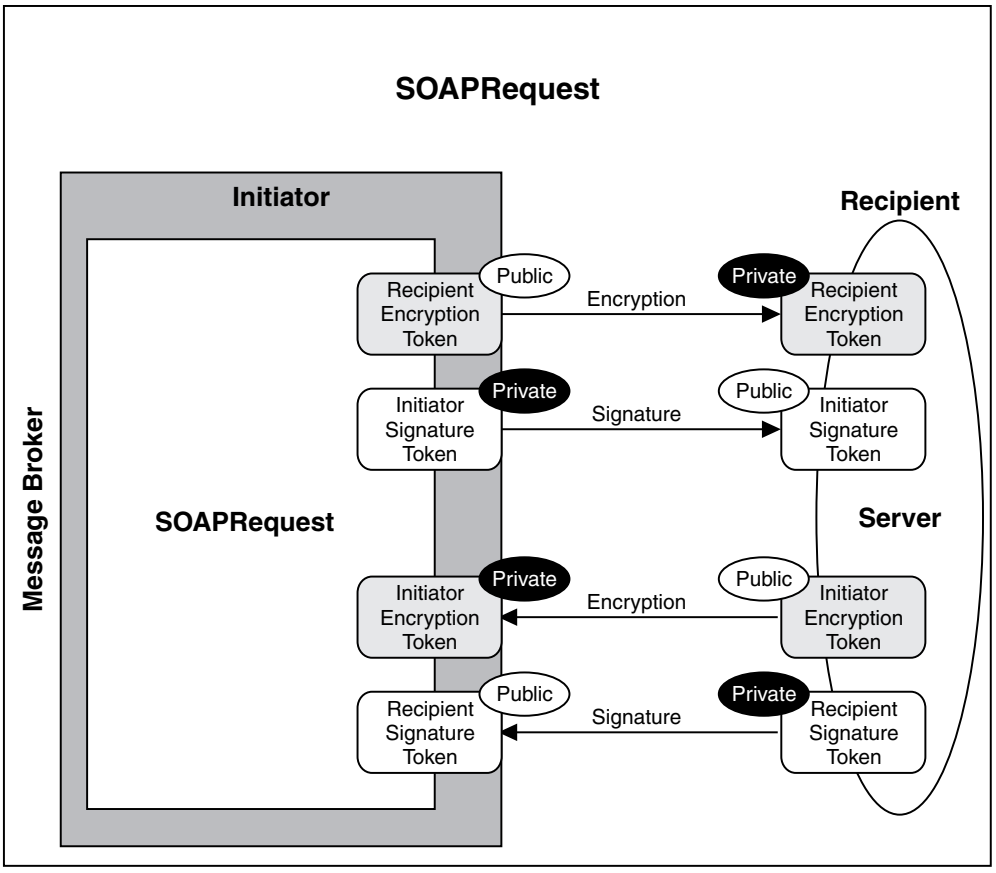
- The initiator uses the broker's public encryption token to encrypt the message, and its own private signature token to sign it.
- The broker uses its own private encryption token to decrypt the message, and the initiator's public signature token to verify the signature.

In the response:

- The broker uses the initiator's public encryption token to encrypt the message, and its own private signature token to sign the message.
- The initiator uses its own private encryption token to decrypt the message, and the broker's public signature token to verify the signature.

SOAPRequest node

This diagram shows the broker acting as an initiator. It uses the SOAPRequest node to make a synchronous request to an external provider (the recipient). Inbound and outbound messages are signed and encrypted. Use of tokens is similar to the example of the asynchronous SOAP nodes, shown earlier.



In the request:

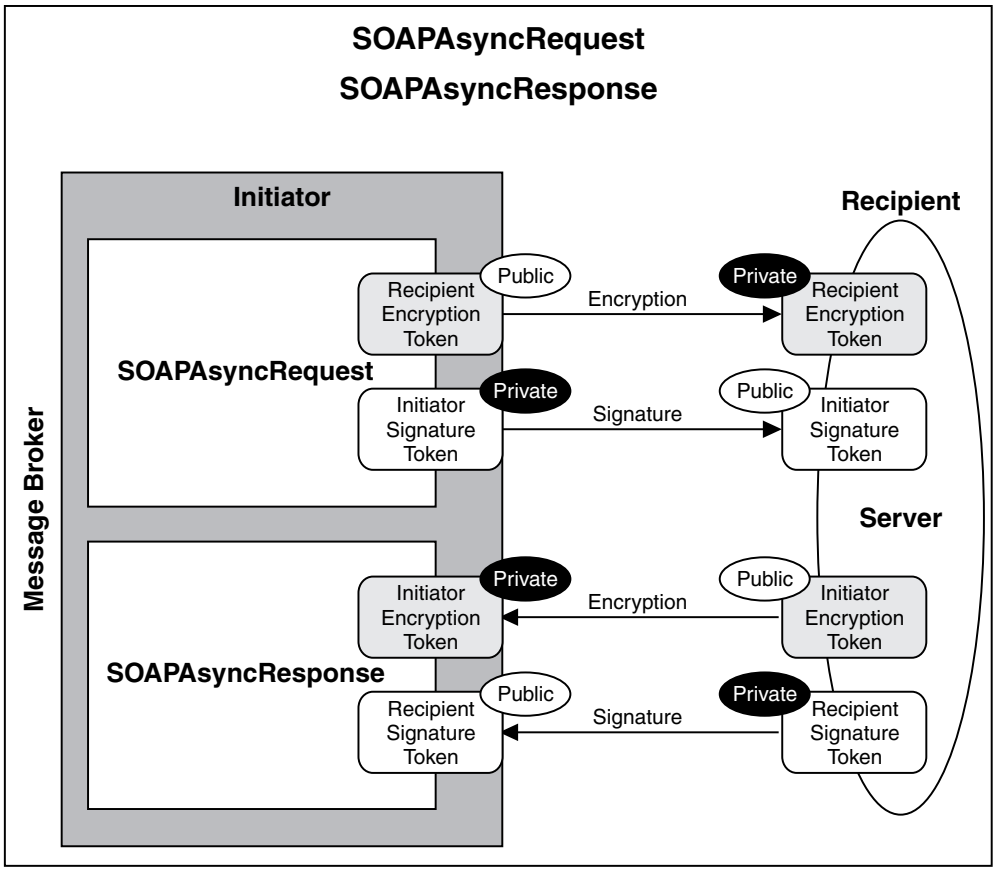
- The broker uses the recipient's public encryption token to encrypt the message, and its own private signature token to sign the message.
- The recipient uses its own private encryption token to decrypt the message, and the broker's public signature token to verify the signature.

In the response:

- The recipient uses the broker's public encryption token to encrypt the message, and its own private signature token to sign the message.
- The broker uses its own private encryption token to decrypt the message, and the initiator's public signature token to verify the signature.

Asynchronous SOAP nodes

This diagram shows the broker acting as an initiator. It uses the asynchronous SOAP nodes to make a request to an external provider (the recipient). Inbound and outbound messages are signed and encrypted.



In the request:

- The broker uses the recipient's public encryption token to encrypt the message, and its own private signature token to sign the message.
- The recipient uses its own private encryption token to decrypt the message, and the broker's public signature token to verify the signature.

In the response:

- The recipient uses the broker's public encryption token to encrypt the message, and its own private signature token to sign the message.
- The broker uses its own private encryption token to decrypt the message, and the initiator's public signature token to verify the signature.

Viewing and setting keystore and truststore runtime properties at broker level

Configure the message broker to refer to a keystore, a truststore, or both, before deploying any message flows that require policy set or bindings for signature, encryption, or X.509 Authentication.

Keystores and truststores are both keystores. They differ only in the way they are used.

Put all private keys and public key certificates (PKC) in the keystore.

Put all trusted root certificate authority (CA) certificates in the truststore. These certificates are used to establish the trust of any inbound public key certificates.

The only supported type of store is Java keystore (JKS).

Each instance of a broker can be configured to refer to one keystore and one truststore.

The following properties of the broker registry component must be defined correctly for policy sets and bindings:

brokerKeystoreFile
The directory and file location of the keystore.

brokerTruststoreFile
The directory and file location of the truststore.

Listing existing broker registry entries:

To display all broker registry values, run the command:

```
mqsireportproperties broker_name -o BrokerRegistry -a
```

This returns entries like these:

```
BrokerRegistry=''
  uuid='BrokerRegistry'
  brokerKeystoreType='JKS'
  brokerKeystoreFile=''
  brokerKeystorePass='brokerKeystore::password'
  brokerTruststoreType='JKS'
  brokerTruststoreFile=''
  brokerTruststorePass='brokerTruststore::password'
  httpConnectorPortRange=''
  httpsConnectorPortRange=''
```

Updating the broker reference to a keystore:

To update the broker reference to a keystore, use the following command:

```
mqsichangeproperties broker_name -o BrokerRegistry
  –n  brokerKeystoreFile
  -v c:\keystore\server.keystore
```

Where `c:\keystore\server.keystore` is the keystore to be referenced.

Updating the broker reference to a truststore:

To update the broker reference to a truststore, use the following command:

```
mqsichangeproperties broker_name -o BrokerRegistry
  –n  brokerTruststoreFile
  -v c:\truststore\server.truststore
```

Where `c:\truststore\server.truststore` is the truststore to be referenced.

Updating the broker with the keystore password:

Keystores and truststores normally require passwords for access. Use the mqsisetdbparms command to add these passwords to the broker runtime component.

```
mqsisetdbparms broker_name
  -n brokerKeystore::password
  -u temp -p pa55word
```

The user ID, which can be any value, is not required to access the keystore.

Updating the broker with the truststore password:

To update the broker with the truststore password, use the following command:

```
mqsisetdbparms broker_name
  -n brokerTruststore::password
  -u temp -p pa55word
```

The user ID, which can be any value, is not required to access the keystore.

Updating the broker with a private key password:

Private keys in the keystore might have their own individual passwords. These can be configured based on the alias name that is specified for the key in the Policy sets and bindings editor. If a key password based on the alias is not found, the keystore password is used. The following command updates the broker with the private key password for the key whose alias is *encKey*.

```
mqsisetdbparms broker_name
  -n brokerTruststore::keypass::encKey
  -u temp -p pa55word
```

The user ID, which can be any value, is not required to access the keystore.

Viewing and setting keystore and truststore runtime properties at execution group level

Configure an execution group to refer to a keystore, a truststore, or both, before deploying any message flows that require policy set or bindings for signature, encryption, or X.509 Authentication.

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

Execution group keystore and truststore runtime property values override equivalent property values on the broker, if any are set.

Keystores can contain two kinds of entries: *keyEntrys* and *trustedCertificateEntries*. If a keystore is used to contain trusted certificates then it is typically referred to as a truststore. WebSphere Message Broker can refer to a keystore and a truststore per execution group. When the broker is encrypting or decrypting it uses entries in its keystore; if it is verifying a signature or performing X.509 authentication it uses entries in its truststore.

The following sample demonstrates the use of viewing and setting keystore and truststore runtime properties at execution group level:

- Address Book

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

Displaying execution group level properties:

To display execution group level properties, run the command:

```
mqsireportproperties broker_name -o ComIbmJVMManager -a -e execution_group
```

Updating the execution group reference to a keystore:

To update the broker reference to a keystore at an execution group level, use the following command:

```
mqsichangeproperties broker_name -e execution_group -o ComIbmJVMManager
  –n  KeystoreFile
  -v c:\keystore\server.keystore,JKS
```

Where `c:\keystore\server.keystore,JKS` is a Java keystore (JKS).

Updating the execution group reference to a truststore:

To update the broker reference to a truststore at an execution group level, use the following command:

```
mqsichangeproperties broker_name -e execution_group -o ComIbmJVMManager
  –n  TruststoreFile
  -v c:\truststore\server.truststore
```

Where `c:\truststore\server.truststore,JKS` is the JKS truststore to be referenced.

Updating the keystore and truststore passwords:

The commands used to update the keystore and truststore passwords at execution group level are the same as those used when setting keystore and truststore runtime properties at broker level.

- To update the broker with the keystore password; see “Updating the broker with the keystore password” on page 770.
- To update the broker with the truststore password; see “Updating the broker with the truststore password” on page 771.
- To update the broker with a private key password; see “Updating the broker with a private key password” on page 771.

To use the default broker password for the keystore, the keystorePass parameter must be blank, or it must be set to `brokerKeystore::password`. To use a password other than the default broker password, use the following commands:

```
mqsichangeproperties broker_name -e execution_group -o ComIbmJVMManager -n keystorePass
-v execution_group::keystorePass

mqsisetdbparms broker_name -n execution_group::keystorePass -u na -p password
```

To use the default broker password for the truststore, the truststorePass parameter must be blank, or it must be set to `brokerTruststore::password`. To use a password other than the default broker password, use the following commands:

```
mqsichangeproperties broker_name -e execution_group -o ComIbmJVMManager -n truststorePass
-v execution_group::truststorePass

mqsisetdbparms broker_name -n execution_group::truststorePass -u na -p password
```

Associating policy sets and bindings with message flows and nodes

Use the Broker Archive editor to associate policy sets and bindings with message flows and nodes, so that they are available to the broker at run time.

Before you start

Use the Policy Sets and Policy Set Bindings editor to create and configure policy sets and bindings.

Associations can be made between policy sets and message flow, or specific nodes. Associations made with a flow apply to all nodes described in the Policy Set and Bindings file. Associations at the flow level are defined as being either for consumer or provider nodes.

An association at the node level overrides any association made at the flow level. You do not enter information about consumer or provider for an association at node level.

1. In the workbench, switch to the Broker Administration perspective.
2. Open the BAR file in the Broker Archive editor.
3. Click the **Manage and Configure** tab.
4. Click the message flow or node that you want to associate with a policy set and binding. The properties that you can configure for the message flow or for the node are displayed in the **Properties** view.
5. If you are configuring a message flow, enter values in the following fields in the **Properties** view, as appropriate:

   Provider Policy Set Bindings

   Provider Policy Set

   Consumer Policy Set Bindings

   Consumer Policy Set
6. If you are configuring a node, enter values in the following fields in the **Properties** view:

   Policy Set

   Policy Set Bindings

For new associations to take effect, the BAR file must be redeployed and the message flows stopped and restarted.

Exporting a policy set and policy binding

Use the mqsireportproperties command to export a policy set and associated binding to a file.

This topic shows how to export policy set *myPolicySet* from broker *myBroker* to a file called *myPolicySet.xml*. The associated binding is *myPolicySetBinding*, which you export to *myPolicySetBinding.xml*.

1. Export the policy set to a file:

   ```
   mqsireportproperties myBroker -c PolicySets -o myPolicySet -n ws-security -p myPolicySet.xml
   ```
2. Export the policy set binding to a file:

   ```
   mqsireportproperties myBroker -c PolicySetBindings -o myPolicySetBinding
      -n ws-security -p myPolicySetBinding.xml
   ```

Make a note of the policy set that is associated to the binding; you will need this information when you import the policy set and binding. This command displays the policy set associated with a binding:

```
mqsireportproperties myBroker -c PolicySetBindings -o myPolicySetBinding -n associatedPolicySet
```

This displays:

```
PolicySetBindings   myPolicySetBinding     associatedPolicySet='myPolicySet'
BIP8071I: Successful command completion.
```

Importing a policy set and policy set binding

Use the mqsichangeproperties command to import a policy set and associated binding.

This topic shows how to import policy set *myPolicySet* to broker *myBroker* from a file called *myPolicySet.xml*. The associated binding is *myPolicySetBinding*, which you import from *myPolicySetBinding.xml*.

1. Create a configurable service for the policy set, if one does not already exist.

   ```
   mqsicreateconfigurableservice myBroker -c PolicySets -o myPolicySet
   BIP8071I: Successful command completion.
   ```

2. Create a configurable service for the policy set binding, if one does not already exist.

   ```
   mqsicreateconfigurableservice myBroker -c PolicySetBindings -o myPolicySetBinding
   BIP8071I: Successful command completion.
   ```

3. Import the policy set.

   ```
   mqsichangeproperties myBroker -c PolicySets -o myPolicySet -n ws-security -p myPolicySet.xml
   ```

4. Import the policy set binding.

   ```
   mqsichangeproperties myBroker -c PolicySetBindings -o myPolicySetBinding
      -n ws-security -p myPolicySetBinding.xml
   ```

5. Change the value of the associatedPolicySet attribute. Set it to the name of the policy set with which this policy set binding was originally associated.

   ```
   mqsichangeproperties myBroker -c PolicySetBindings -o myPolicySetBinding
      -n associatedPolicySet -v myPolicySet
   ```

Message flow security and security profiles

WebSphere Message Broker provides a security manager for implementing message flow security, so that end-to-end processing of a message through a message flow is secured based on an identity carried in that message instance.

For details of the supported external providers and the operation of the message flow security manager, see Message flow security . For information about the token types that are supported by the SOAP nodes and by external security providers, see Identity.

When the message flow is a Web service implemented by using "SOAP nodes" on page 742 and the identity is to be taken from the "WS-Security" on page 759 SOAP headers, the SOAP nodes are the Policy Enforcement Point (PEP) and the external provider defined by the Security profiles is the Policy Decision Point (PDP).

The following configuration is required to implement message flow security based on an identity carried in `WS_Security` tokens.

- "Policy sets" on page 764 define the type of tokens used for the identity.
  - To work with a Username and Password identity, configure the policy and binding for Username token "Authentication" on page 762.
  - To work with a X.509 Certificate identity, configure the policy and binding for X.509 certificate token "Authentication" on page 762.

    In the Policy Set Binding, set the X.509 certificate Authentication Token certificates mode to Trust Any. You set it this way (and not to Trust Store) so that the certificate is passed to the security provider defined by the Security Profile. Setting it to Trust Store will cause the certificate to be validated in the local Broker Trust Store. For more details, see Policy Sets and Policy Set Bindings editor: Authentication and Protection Tokens panel.

- The message flow security operation and external provider are defined by the Security profiles

As an alternative to message flow security and an external PDP, the broker's truststore can be used as a local PDP for X.509 certificate authentication. For WS-Security signing and encryption using only the local broker capability, you must configure the broker's truststore. For details, see "Viewing and setting keystore and truststore runtime properties at broker level" on page 769, or "Viewing and setting keystore and truststore runtime properties at execution group level" on page 771.

WS-Security capabilities

Web service security capabilities are supported by the broker.

Web service security mechanisms are defined by OASIS standards. See OASIS Standard for WS-Security Specification

For information about the token profile standards, see:

- OASIS Web Services Security Username Token Profile
- OASIS Web Services Security X.509 Certificate Token Profile

For more information about using the token profiles, see the following topics:

- "Username token capabilities"
- "X.509 certificate token capabilities" on page 777

Username token capabilities

This topic describes WS-Security username token capabilities of the broker.

For details of using WS-Security username token, see the following capabilities:

- "Username token capabilities for encryption, decryption, signing, and verifying"
- "Username token capabilities for authentication and authorization"
- "Username token capabilities for identity mapping" on page 776
- "Username token capabilities for extraction and propagation" on page 776

Username token capabilities for encryption, decryption, signing, and verifying:

For Web services, you cannot complete encryption, decryption, signing, and verification by using username tokens.

The username token is not applicable, or supported, for the following in any configuration or direction:

- Encryption
- Decryption
- Signing
- Verification

Username token capabilities for authentication and authorization:

For Web services, you can complete authentication and authorization by using a username token.

The username token "Authentication" on page 761 and Authorization is supported only in the following configuration:

Capability

- Authenticate
- Authorize

Policy Enforcement Point (PEP) and direction

- In (provider)

  "SOAPInput node" on page 1194

  Configured with a security policy and binding that defines that a username token is present for authentication; see "Authentication" on page 762. You can use the default policy and binding WSS10Default; see "Default policy set and bindings" on page 765.

  Configured with a security profile defining the Policy Decision Point (PDP); see the PDP section that follows.

Trust Store or PDP

- LDAP

  Configured by using an LDAP security profile specifying authentication, authorization, or both; see Creating a security profile for LDAP.

- TFIM V6.1

  Configured by using a TFIM security profile specifying authentication, authorization or both; see Creating a security profile for TFIM V6.1.

Username token capabilities for identity mapping:

For Web services, you can map an identity by using a username token.

Identity mapping from a username identity token to a mapped username identity token is supported only in the following configurations:

Capability

- Identity mapping

Policy Enforcement Point (PEP) and direction

- In (provider)

  "SOAPInput node" on page 1194

  Configured with a security policy and binding that defines that a username taken is present. You can use the default policy and binding WSS10Default; see "Default policy set and bindings" on page 765.

  Configured with a security profile defining the external Policy Decision Point (PDP); see the PDP section that follows.

Trust store or PDP

- TFIM V6.1

  Configured by using a TFIM security profile that specifies identity mapping; see Creating a security profile for TFIM V6.1.

Username token capabilities for extraction and propagation:

This topic describes broker capability for extraction, propagation, or both using a username token in Web services.

The extraction of username into the Properties folder source Identity fields, is supported in the following configurations:

Capability
- Extraction

Policy Enforcement Point (PEP) and direction
- In (provider)

  "SOAPInput node" on page 1194

  Configured with a security policy and binding which defines that a username taken is present. You can use the default policy and binding WSS10Default; see "Default policy set and bindings" on page 765.

  Configured with a security profile that defines propagation; see Creating a security profile

The propagation of a username token into the SOAP WS-Security header, from the token present in either the mapped or the source identity fields in the properties folder, is supported in the following configuration. See Identity.

Capability
- Propagate

Policy Enforcement Point (PEP) and direction
- Out (consumer)

  "SOAPRequest node" on page 1204

  "SOAPAsyncRequest node" on page 1172

  Configured with a security profile that defines propagation; for example, Default Propagation. See Security profiles

X.509 certificate token capabilities

Various WS-Services Security X.509 certificate token profile standards are supported by WebSphere Message Broker.

For details of using the X.509 certificates, see the following capabilities:
- "X.509 certificate token capabilities for encryption"
- "X.509 certificate token capabilities for decryption" on page 778
- "X.509 certificate token capabilities for signing" on page 778
- "X.509 certificate token capabilities for verifying" on page 779
- "X.509 certificate token capabilities for authentication" on page 779
- "X.509 certificate token capabilities for authorization" on page 780
- "X.509 certificate token capabilities for identity mapping" on page 780
- "X.509 certificate token capabilities for extraction and propagation" on page 780

X.509 certificate token capabilities for encryption:

For Web services, you can complete encryption by using an X.509 certificate token.

X.509 certificate token encryption for providing message "Confidentiality" on page 762 on outgoing SOAP messages from the broker is supported in the following configurations:

Capability

- Encrypt (by using a partner public key)

Policy Enforcement Point (PEP) and direction

- Out (consumer)

  "SOAPRequest node" on page 1204

  "SOAPAsyncRequest node" on page 1172
- Out (provider)

  "SOAPReply node" on page 1202

  Configured with a policy set and binding defining the message "Confidentiality" on page 763.

Trust Store or Policy Decision Point (PDP)

- Broker Truststore; for more details, see "Viewing and setting keystore and truststore runtime properties at broker level" on page 769.

Encryption is not supported with external PDPs such as TFIM or LDAP.

X.509 certificate token capabilities for decryption:

For Web services, you can complete decryption by using an X.509 certificate token.

X.509 certificate token decryption for incoming SOAP message "Confidentiality" on page 762 is supported in the following configurations:

Capability

- Decrypt (by using a broker private key)

Policy Enforcement Point (PEP) and direction.

- In (provider)

  "SOAPInput node" on page 1194
- In (consumer)

  "SOAPRequest node" on page 1204

  "SOAPAsyncResponse node" on page 1181

  Configured with a policy set and binding defining the message "Confidentiality" on page 763.

Trust Store or Policy Decision Point (PDP).

- Broker Truststore; for details, see "Viewing and setting keystore and truststore runtime properties at broker level" on page 769.

Decryption is not supported with external PDPs such as TFIM or LDAP.

X.509 certificate token capabilities for signing:

For Web services, you can use an X.509 certificate token for signing.

X.509 certificate token signing for outgoing SOAP message "Integrity" on page 762 is supported in the following configurations:

Capability

- Sign (by using a broker private key)

Policy Enforcement Point (PEP) and direction

- Out (consumer)

  "SOAPRequest node" on page 1204

  "SOAPAsyncRequest node" on page 1172
- Out (provider)

  "SOAPReply node" on page 1202

  Configured with a policy set and binding defining the message "Integrity" on page 764.

Trust Store or Policy Decision Point (PDP)

- Broker Truststore; for details, see "Viewing and setting keystore and truststore runtime properties at broker level" on page 769.

Signing is not supported with an external PDP such as TFIM or LDAP.

X.509 certificate token capabilities for verifying:

For Web services, you can verify a signing by using an X.509 certificate token profile.

X.509 certificate token verification of the "Integrity" on page 762 of a signed incoming SOAP message is supported in the following configurations:

Capability

- Verify signature (by using a partner public key)

Policy Enforcement Point (PEP) and direction

- In (provider)

  "SOAPInput node" on page 1194
- In (consumer)

  "SOAPRequest node" on page 1204

  "SOAPAsyncResponse node" on page 1181

  Configured with a policy set and binding defining the message "Integrity" on page 764

Trust Store or Policy Decision Point (PDP)

- Broker Trust store; for details, see "Viewing and setting keystore and truststore runtime properties at broker level" on page 769.

  Signature verification is not supported with an external PDP, such as TFIM or LDAP.

X.509 certificate token capabilities for authentication:

For Web services, you can complete authentication by using an X.509 certificate token.

The X.509 certificate token "Authentication" on page 761 of an incoming SOAP message is supported in the following configurations:

Capability

- Authenticate

Policy Enforcement Point (PEP) and direction

- In (provider)

  "SOAPInput node" on page 1194

  Configured with a policy set and binding defining the certificate "Authentication" on page 762.

  Optionally configured with a security profile defining an external Policy Decision Point (PDP); see the PDP section that follows.

Trust Store or PDP

- Broker Trust store; for details, see "Viewing and setting keystore and truststore runtime properties at broker level" on page 769.
- TFIM V6.1

  Configured by using a TFIM security profile specifying authentication; for details, see Creating a security profile for TFIM V6.1.

Certificate authentication with an external LDAP PDP is not supported.

X.509 certificate token capabilities for authorization:

The X.509 certificate token is not supported for authorization in any configuration or direction.

X.509 certificate token capabilities for identity mapping:

For Web services, you can map an identity by using an X.509 certificate token.

The broker supports Identity mapping from an X.509 certificate token in an incoming SOAP message header to username tokens in the following configurations:

Capability

- Identity mapping

Policy Enforcement Point (PEP) and direction

- In (provider)

  "SOAPInput node" on page 1194

  Configured with a policy set and binding defining the certificate "Authentication" on page 762.

  Configured with a security profile defining an external Policy Decision Point (PDP); see the PDP section that follows.

Trust Store or PDP

- TFIM V6.1

  Configured by using a TFIM security profile specifying identity mapping; for details, see Creating a security profile for TFIM V6.1.

Identity mapping is not supported with LDAP, or at outbound nodes.

Username tokens only can be propagated.

X.509 certificate token capabilities for extraction and propagation:

This topic describes broker Web services capability for extraction and propagation X.509 certificate token.

The broker does not support propagation of an X.509 certificate.

The X.509 certificate token extraction is supported in the following configurations:

Capability
- Extraction

Policy Enforcement Point (PEP) and direction
- In (provider)

  "SOAPInput node" on page 1194

  Configured with a policy set and binding defining the X.509 certificate is present; see "Implementing WS-Security" on page 762.

  Configured with a security profile defining propagation; see the Security profiles.

WebSphere Service Registry and Repository

The WebSphere Service Registry and Repository (WSRR) is a central repository of entities. A wide range of entities can be stored and retrieved, including user-defined concepts and definitions related specifically to Web services, such as WSDL services, service interfaces, and associated policies.

WebSphere Message Broker Version 6.1.0.4 and later supports only WSRR 6.1 or later release of version 6.

You can configure a message flow to dynamically retrieve resources from WSRR at run time, and to use and expose those resources in the message flow. You can therefore defer the decision about which resources you want to use until run time, rather than making the decision at deployment time.

WSRR has specific support for many of the document types associated with Web services, including generic XML documents, WSDL, and SCDL. For example, when you load a WSDL document into WSRR it also identifies and stores its individual logical components, such as the service and port type.

Use the WSRR nodes (the RegistryLookup and EndpointLookup nodes) to create message flows that retrieve data dynamically from WSRR. Data is retrieved according to search criteria defined by node properties, possibly supplemented or overridden by local environment definitions. The retrieved data is placed in the local environment tree, which makes the data available to subsequent nodes. The input message received by the node is propagated to the output terminal unchanged.

Use the RegistryLookup node to submit generic queries to WSRR. Entities returned by the query are stored in the ServiceRegistry output tree in the local environment. You can also specify that details of the relationships between the returned entities and other entities that they reference are represented in the ServiceRegistry output tree.

Use the EndpointLookup node to submit queries for Web service endpoints. This node is tailored to retrieve WSDL port definitions that implement a specified WSDL portType. The details of service endpoints that match the specified criteria

are placed in the ServiceRegistry output tree in the local environment. If the node is configured to return a single matching service endpoint the Web service URL destination used by the SOAP and HTTP request nodes is also overridden in the local environment. If the node is configured to return all matching service endpoints, the local environment is not set up automatically for the SOAP and HTTP request nodes. In this case the local environment tree might contain data for multiple service endpoints, and the message flow should interpret and use this information.

Set the configuration parameters for the WSRR nodes to specify how WebSphere Message Broker interfaces with your WSRR server.

- Use the **connectionTimeout** parameter to set a system-wide connection timeout for queries that are issued by the EndpointLookup and RegistryLookup nodes. If a query result is not returned from the WSRR server before the connection timeout period expires, the configured error handling is invoked.
- Use the **needCache** parameter to enable the WebSphere Message Broker WSRR cache. The cache is used to store results from queries that are issued by the EndpointLookup and RegistryLookup nodes.
- If the WebSphere Message Broker WSRR cache is enabled, use the **timeout** parameter to set a system-wide cache timeout. The cache timeout controls how long results from queries that are stored in the cache are used before the query is reissued.

If your WebSphere Message Broker application is running on Sun Solaris 10 on SPARC, you might need to increase the number of file descriptors. If there are not enough file descriptors, you might fail to get a response to a WSRR query within the configured timeout period. In addition, the service trace might contain one or more java.lang.SecurityExceptions related to com.ibm.ws.tcp.channel.impl.ChannelSelector, and an abend file might be produced.

The topics in this section provide further information about working with WSRR:

- "Configuration parameters for the WebSphere Service Registry and Repository nodes"
- "Displaying the configuration parameters for the WebSphere Service Registry and Repository nodes" on page 784
- "Changing the configuration parameters for the WebSphere Service Registry and Repository nodes" on page 785
- "Accessing a secure WebSphere Service Registry and Repository" on page 786
- "Caching artifacts from the WebSphere Service Registry and Repository" on page 789
  - "Setting up cache notification" on page 790
- "Dynamically defining the search criteria" on page 791
- "EndpointLookup node" on page 940
- "EndpointLookup node output" on page 793
- "RegistryLookup node" on page 1132
- "RegistryLookup node output" on page 795

Configuration parameters for the WebSphere Service Registry and Repository nodes

These parameters affect the brokers configuration when interfacing with WebSphere Service Registry and Repository (WSRR).

The following table describes the parameters used by the mqsichangeproperties command to configure the `DefaultWSRR` object of the Service Registries configurable service.

Configuration Setting Name	Default value	Description

endpointAddress	`http://`*`hostname`*`:9080/WSRR6_1/services/WSRRCoreSDOPort`
	or for WSRR 6.2:
	`http://`*`hostname`*`:9080/WSRR6_2/services/WSRRCoreSDOPort`
	or for WSRR 6.3:
	`http://`*`hostname`*`:9080/WSRR6_3/services/WSRRCoreSDOPort`
		Endpoint of the WSRR server. Where *hostname:port* is host address and port of your WSRR server. To connect to a secure WSRR see "Accessing a secure WebSphere Service Registry and Repository" on page 786

timeoutConnection	180	The WSRR connection timeout period in seconds. The default value is 180 seconds, which is 3 minutes. Set a value that is sufficiently long for complex queries to complete.

The following table describes the parameters for Cache that are configured by using the mqsichangeproperties command.

Configuration Setting Name	Default value	Description
needCache	True	Enable WebSphere Message Broker WSRR cache.
timeout	100000000	The timeout value for the cache. The cache expiry time in milliseconds. (The default value of 100000000 milliseconds is approximately 27.8 hours).
predefinedCacheQueries	None	A list of queries separated by semicolons with which to initialize the WebSphere Message Broker WSRR cache. This query is defined by the WSRR query language.

The following table describes the parameters for Cache Notification that are configured by using the mqsichangeproperties command.

Configuration Setting Name	Default value	Description
enableCacheNotification	False	Enable WebSphere Message Broker WSRR Cache Notification.
refreshQueriesAfterNotification	True	When a notification is received from WSRR, if refreshQueriesAfterNotification is set to True the cache is updated with the new version of the object immediately; if False, the cache will be updated on the next request.

Configuration Setting Name	Default value	Description
connectionFactoryName	jms/SRConnectionFactory	The name of the WSRR WebSphere Application Server JMS provider JMS connection factory for Cache Notification.
initialContextFactory	com.ibm.websphere.naming. WsnInitialContextFactory	The name of the WSRR WebSphere Application Server JMS provider JMS context factory for Cache Notification.
locationJNDIBinding	`iiop://host.name:2809/`	The URL to the WebSphere Application Server JMS provider JNDI bindings. Where *host.name* is variable.
subscriptionTopic	jms/SuccessTopic	The topic name used to receive WebSphere Application Server JMS provider Cache Notification.

For information on accessing a secure WSRR server see "Accessing a secure WebSphere Service Registry and Repository" on page 786

Displaying the configuration parameters for the WebSphere Service Registry and Repository nodes

Use the mqsireportproperties command to display all of the configuration parameters of the default WebSphere Service Registry and Repository (WSRR) profile, `DefaultWSRR`.

The `DefaultWSRR` is a Service Registries configurable service that is supplied for each broker, see Configurable services properties.

To display the configuration parameters of the default WSRR profile `DefaultWSRR`, complete the following steps:

1. Ensure that the broker is running. If it is not, use the mqsistart command to start it.
2. Enter the following command (where WBRK_BROKER is the name of your broker):

   ```
   mqsireportproperties WBRK_BROKER -c ServiceRegistries -o DefaultWSRR -r
   ```

   where:

   -c specifies the configurable service (in this case, ServiceRegistries)
   -o specifies the name of the object (in this case, DefaultWSRR)
   -r specifies that all property values of the object are displayed, including the child values if appropriate.

The command produces a response similar to this:

```
ReportableEntityName=''
ServiceRegistries
  DefaultWSRR=''
    connectionFactoryName = 'jms/SRConnectionFactory'
    connectionTimeout = '120'
    endpointAddress = 'http://localhost:9080/WSRR6_3/services/WSRRCoreSDOPort'
    initialContextFactory = 'com.ibm.websphere.naming.WsnInitialContextFactory'
    locationJNDIBinding = 'iiop://localhost:2809/'
    needCache = 'true'
```

```
predefinedCacheQueries = ''
refreshQueriesAfterNotification = 'true'
subscriptionTopic = 'jms/SuccessTopic'
timeout = '10000000'
```

Changing the configuration parameters for the WebSphere Service Registry and Repository nodes

Use the mqsichangeproperties command to change the configuration parameters of the DefaultWSRR configurable service.

DefaultWSRR is a configurable service object that is supplied for each broker, it defines the WebSphere Service Registry and Repository (WSRR) configuration parameters. DefaultWSRR has a configurable service type of ServiceRegistries.

For details about configuration parameters that affect WSRR use, see "Configuration parameters for the WebSphere Service Registry and Repository nodes" on page 782.

To update the configuration parameters of the DefaultWSRR configurable service perform the following steps:

1. Ensure that the broker is running. If it is not, use the mqsistart command to start it.
2. Enter the following command to change the endpointAddress value and point to your WebSphere Service Registry and Repository server:

   ```
   mqsichangeproperties WBRK_BROKER -c ServiceRegistries -o DefaultWSRR
   -n endpointAddress
   -v http://localhost:9080/WSRR6_1/services/WSRRCoreSDOPort
   ```

   where:

   -c specifies the configurable service type
   (in this case, ServiceRegistries)
   -o specifies the name of the configurable service object
   (in this case, DefaultWSRR)
   -n specifies the names of the properties to be changed
   (in this case, endpointAddress)
   -v specifies the values of properties defined by the **-n** parameter
   (in this case,
   http://localhost:9080/WSRR6_1/services/WSRRCoreSDOPort)

   For WSRR 6.3, the address becomes: http://localhost:9080/WSRR6_3/services/WSRRCoreSDOPort. For WSRR 7.0 with WSRR 6.1 compatibility mode enabled, the address is: http://localhost:9080/WSRR6_1/services/WSRRCoreSDOPort.
3. (Optional) Enter the following command to change the cache timeout value:

   ```
   mqsichangeproperties WBRK_BROKER -c ServiceRegistries -o DefaultWSRR
   -n timeout -v 3600000
   ```

   where:

   -c specifies the configurable service type
   (in this case, ServiceRegistries)
   -o specifies the name of the configurable service object
   (in this case, DefaultWSRR)
   -n specifies the names of the properties to be changed
   (in this case, timeout)

-v specifies the values of properties defined by the **-n** parameter
(in this case, 3600000 milliseconds to provide WSRR cache expiry timeout of 1 hour)

4. (Optional) Enter the following command to change the connectionTimeout value:

```
mqsichangeproperties WBRK_BROKER -c ServiceRegistries -o DefaultWSRR
-n connectionTimeout -v 240
```

where:

-c specifies the configurable service type
(in this case, `ServiceRegistries`)
-o specifies the name of the configurable service object
(in this case, `DefaultWSRR`)
-n specifies the names of the properties to be changed
(in this case, `connectionTimeout`)
-v specifies the values of properties defined by the **-n** parameter
(in this case, 240 seconds to provide connection timeout for WSRR queries of 4 minutes)

5. (Optional) Enter the following command to preload the cache at broker startup with the results of specific queries:

```
mqsichangeproperties WBRK_BROKER -c ServiceRegistries -o DefaultWSRR
-n predefinedCacheQueries
-v "//*[@name=&apos;ConceptA1&apos;]"
```

where:

-c specifies the configurable service type
(in this case, `ServiceRegistries`)
-o specifies the name of the configurable service object
(in this case, `DefaultWSRR`)
-n specifies the names of the properties to be changed
(in this case, `predefinedCacheQueries`)
-v specifies the values of properties defined by the **-n** parameter
(in this case a simple full depth WSRR XPath query on the entity `ConceptA1`, `"//*[@name=&apos;ConceptA1&apos;]"`).

Note that single quotation marks in the WSRR query must be replaced by `&apos;)`

Multiple queries can be specified by delimiting them with ';'

For example, to perform a full depth query on the entities named `ConceptA1` and `ConceptB2` use:

```
-v "//*[@name=&apos;ConceptA1&apos;];//*[@name=&apos;ConceptB2&apos;]"
```

Individual queries can use the full power of the WSRR query language:

```
-v "/WSRR/WSDLService/ports[binding/portType
[@name=&apos;DemoCustomer&apos;
and @namespace=&apos;http://demo.sr.eis.ibm.com&apos;]]"
```

6. Restart the broker, by using the mqsistop command to stop the broker, followed by the mqsistart command to start it.

Accessing a secure WebSphere Service Registry and Repository

To access a secure WebSphere Service Registry and Repository (WSRR), you must set the configuration parameters by using the mqsichangeproperties command.

You must connect over HTTPS, not HTTP, which is specified in the endpointAddress configuration parameter of the default WSRR profile, `DefaultWSRR`.

For more information about the endpointAddress configuration parameter, see "Configuration parameters for the WebSphere Service Registry and Repository nodes" on page 782.

To access a secure WebSphere Service Registry and Repository, enter the following sequence of commands:

1. Ensure that the broker is running. If it is not, use the mqsistart command to start it.
2. Display the configuration parameters of the BrokerRegistry by using the following command:

   ```
   mqsireportproperties WBRK_BROKER -o BrokerRegistry -r
   ```

   where:

   -o specifies the name of the object (in this case, BrokerRegistry)
   -r specifies that all property values of the object are displayed, including the child values if appropriate.
3. Change the endpointAddress configuration parameter to specify https and the secure port for the `DefaultWSRR` of the ServiceRegistries configurable service by using the following command:

   ```
   mqsichangeproperties WBRK_BROKER -c ServiceRegistries -o DefaultWSRR
   -n endpointAddress
   -v https://localhost:9443/WSRR6_1/services/WSRRCoreSDOPort
   ```

   where:

   -c specifies the configurable service (in this case, `ServiceRegistries`)
   -o specifies the name of the object (in this case, `DefaultWSRR`)
   -n specifies the names of the properties to be changed (in this case, `endpointAddress`)
   -v specifies the values of properties defined by the **-n** parameter (in this case, `https://localhost:9443/WSRR6_1/services/WSRRCoreSDOPort`)

   If you use WSRR 6.3 this value becomes `https://localhost:9443/WSRR6_3/services/WSRRCoreSDOPort`. If you use WSRR 7.0 with WSRR 6.1 compatibility mode enabled, this value is `https://localhost:9443/WSRR6_1/services/WSRRCoreSDOPort`.
4. The broker keystore must be configured to contain your WSRR server certificate keys; for a discussion of digital certificates, see Digital certificates. Obtain these certificate keys from the installation of the WebSphere Application Server that hosts your WSRR server. The broker uses a single keystore, so, if your broker also implements WS-Security, HTTPS, or SSL secured WebSphere MQ, you might need to merge the provided keys into an existing keystore file. The broker keystore is configured by using the mqsichangeproperties command. Change the brokerKeystoreFile configuration parameters for the broker by using the following command:

   ```
   mqsichangeproperties WBRK_BROKER -o BrokerRegistry
   -n brokerKeystoreFile -v C:\WSRR\SSL\ClientKeyFile.jks
   ```

   where:

-o specifies the name of the object (in this case, `BrokerRegistry`)
-n specifies the names of the properties to be changed (in this case, `brokerKeystoreFile`)
-v specifies the values of properties defined by the **-n** parameter (in this case, `C:\WSRR\SSL\ClientKeyFile.jks`)

5. The broker truststore must be configured to contain signer certificates for your WSRR server. As described previously for the keystore, the broker uses a single truststore, therefore certificates might need to be merged into an existing truststore file. The broker truststore is configured by using the mqsichangeproperties command. Change the brokerTruststoreFile configuration parameters for the broker by using the following command:

   ```
   mqsichangeproperties WBRK_BROKER -o BrokerRegistry
   -n brokerTruststoreFile -v C:\WSRR\SSL\ClientTrustFile.jks
   ```

   where:

   -o specifies the name of the object (in this case, `BrokerRegistry`)
   -n specifies the names of the properties to be changed (in this case, `brokerTruststoreFile`)
   -v specifies the values of properties defined by the **-n** parameter (in this case, `C:\WSRR\SSL\ClientTrustFile.jks`)
6. Stop the broker by using the mqsistop command.
7. Set the WebSphere Application Server user name and password by using the following command:

   ```
   mqsisetdbparms WBRK_BROKER -n DefaultWSRR::WSRR -u wasuser -p waspass
   ```

   where:

   -n specifies the name of the data source (in this case, `DefaultWSRR::WSRR`)
   -u specifies the user ID to be associated with this data source (in this case, `wasuser`)
   -p specifies the password to be associated with this data source (in this case, `waspass`)
8. Set the brokerKeystore user name and password by using the following command:

   ```
   mqsisetdbparms WBRK_BROKER -n brokerKeystore::password -u dummy -p WebAS
   ```

   where:

   -n specifies the name of the data source (in this case, `brokerKeystore::password`)
   -u specifies the user ID to be associated with this data source (in this case, `dummy`)
   -p specifies the password to be associated with this data source (in this case, `WebAS`)
9. Set the brokerTrustStore user name and password by using the following command:

   ```
   mqsisetdbparms WBRK_BROKER -n brokerTruststore::password -u dummy
   -p WebAS
   ```

   where:

   -n specifies the name of the data source (in this case, `brokerTruststore::password`)
   -u specifies the user ID to be associated with this data source (in this case, `dummy`)
   -p specifies the password to be associated with this data source (in this case, `WebAS`)

10. If you are connecting to a secure WebSphere Application Server, and you want to use Cache Notification, you must use a valid user ID and password for the JMS cache notification service. To set the user ID and password follow these steps:
    a. Stop the broker by using the mqsistop command.
    b. Issue the mqsisetdbparms command to set up your user ID and password. For example:

```
mqsisetdbparms WBRK_BROKER -n jms::DefaultWSRR@jms/SRConnectionFactory
-u <userid> -p <password>
```

    where:

    -n specifies the name of the data source (in this case, `jms::DefaultWSRR@jms/SRConnectionFactory`)
    -u specifies the user ID to be associated with this data source (in this case, *<userid>*)
    -p specifies the password to be associated with this data source (in this case, *<password>*)
11. Restart the broker by using the mqsistop command to stop the broker, followed by the mqsistart command to start it.

Caching artifacts from the WebSphere Service Registry and Repository

WebSphere Message Broker saves the data it retrieves from the WebSphere Service Registry and Repository (WSRR) in a local cache.

The WSRR nodes, EndpointLookup and RegistryLookup, can retrieve data that was stored in the Broker WSRR cache by a previous query, improving performance and message throughput. The first occurrence of each query is always sent to WSRR. By default, this activity occurs when a WSRR node first issues a specific query, although it is possible to pre-populate the cache, when the broker starts, by using the queries described here.

Configuring the WSRR Cache

The cache is configured individually for each broker by using the mqsichangeproperties command; for details, see Configurable services properties.

You can configure the cache in the following ways:

- **Disabling the cache**

  Disable the cache by setting the needCache parameter to false. By default, the cache is enabled, but the WSRR nodes can operate without the cache. If the cache is disabled, every query that is issued by the node is sent to WSRR, ensuring that the results of the query always reflect the current contents of the registry. This activity can affect performance.
- **Preloading the cache**

  Preload the cache by setting the predefinedCacheQueries parameter. By default, no items are preloaded in the cache and the first occurrence of every query is sent to WSRR. You can specify predefined queries that are executed when the broker starts, or when a message flow containing WSRR flows is first deployed, populating the cache for use by subsequent WSRR nodes. By specifying predefined queries, performance might be affected at startup, rather than on the first occurrence of a query at run time.

- **Changing the cache expiry timeout value**

  Change the cache expiry timeout value by setting the timeout parameter. The cached results of a query are discarded after the specified time has elapsed. The next occurrence of the query is sent to WSRR and the new result is entered in the cache. If the contents of the registry are likely to change frequently, you can specify a shorter expiry timeout value so that changes are picked up quicker. This activity affects performance as more queries are sent to WSRR.
- **Enabling cache notification**

  Enable cache notification by setting the enableCacheNotification parameter to true and by setting the initialContextFactory and locationJNDIBinding properties appropriately for your WSRR server. By default, cache notification is disabled. Cache notification is a more flexible method than expiry timeout for refreshing cached data because it allows individual WSRR entities to be refreshed at the time they are modified in WSRR.

  If cache notification is enabled, the cache subscribes to events occurring in WSRR and is notified when an object is updated or deleted in WSRR. The object is discarded from the cache. If the refreshQueriesAfterNotification parameter is set to true, the cache is updated with the new version of the object immediately. If the refreshQueriesAfterNotification parameter is set to false, the cache is updated the next time a relevant query is issued by a WSRR node.

Setting up cache notification

Use the mqsichangeproperties command to enable cache notification, so that the cache is notified of events occurring in WebSphere Service Registry and Repository (WSRR).

WSRR publishes notification events by using WebSphere Application Server. Cache notification allows the cache to subscribe to these events.

To enable cache notification complete the following steps to change the relevant properties on the configurable service `DefaultWSRR`, and to add a user ID and password if you are connecting to a secure WebSphere Application Server:

1. Ensure that the broker is running. If it is not, use the mqsistart command to start it.
2. Issue the mqsichangeproperties command to change the enableCacheNotification property to true. For example:

   ```
   mqsichangeproperties WBRK_BROKER -c ServiceRegistries -o DefaultWSRR
   -n enableCacheNotification -v true
   ```

   where:

   -c specifies the configurable service (in this case, ServiceRegistries)
   -o specifies the name of the object (in this case, DefaultWSRR)
   -n specifies the names of the properties to be changed (in this case, enableCacheNotification)
   -v specifies the values of properties defined by the **-n** parameter (in this case, true)
3. Issue the mqsichangeproperties command to change the locationJNDIBinding property to the value that you require for your WSRR server. For example:

   ```
   mqsichangeproperties WBRK_BROKER -c ServiceRegistries -o DefaultWSRR
   -n locationJNDIBinding -v iiop://localhost:2809/
   ```

   where:

-c specifies the configurable service (in this case, ServiceRegistries)
-o specifies the name of the object (in this case, DefaultWSRR)
-n specifies the names of the properties to be changed
(in this case, locationJNDIBinding)
-v specifies the values of properties defined by the **-n** parameter
(in this case, iiop://localhost:2809/)

4. If you are connecting to a secure WebSphere Application Server you must use a user ID and password. To set the user ID and password follow these steps:
   a. Stop the broker by using the mqsistop command.
   b. Issue the mqsisetdbparms command to set up your user ID and password. For example:

      ```
      mqsisetdbparms WBRK_BROKER -n jms::DefaultWSRR@jms/SRConnectionFactory
      -u <userid> -p <password>
      ```

      where:

      -n specifies the name of the data source
      (in this case, jms::DefaultWSRR@jms/SRConnectionFactory)
      -u specifies the user ID to be associated with this data source
      (in this case, <*userid*>)
      -p specifies the password to be associated with this data source
      (in this case, <*password*>)
   c. Restart the broker by using the mqsistart command.

Dynamically defining the search criteria

You can use the RegistryLookup and EndpointLookup nodes to issue WebSphere Service Registry and Repository (WSRR) queries specified in the local environment.

The RegistryLookup and EndpointLookup nodes issue WSRR queries at run time and save the resulting data in the local environment. You can specify the queries at design time by using node properties to define the search criteria. Both nodes require at least one query property to be defined before you can deploy the message flow. However, you can specify the search criteria at run time in the local environment, either supplementing or overriding the node properties.

The following table defines the local environment overrides for WSRR queries. These fields must be set in OutputLocalEnvironment.ServiceRegistryLookupProperties by a preceding transformation node, such as a Compute node.

Setting	Description
Name	This setting overrides the Name property on the node; for example, with an ESQL Compute node: `SET OutputLocalEnvironment.ServiceRegistryLookupProperties.Name = 'DemoCustomer';`
Namespace	This setting overrides the Namespace property on the node; for example: `SET OutputLocalEnvironment.ServiceRegistryLookupProperties.Namespace = 'http://mb.sr.eis.ibm.com';`
Version	This setting overrides the Version property on the node; for example: `SET OutputLocalEnvironment.ServiceRegistryLookupProperties.Version = '1.0';`

Setting	Description
Template	This setting overrides the Template property on the RegistryLookup node; for example: `SET OutputLocalEnvironment.ServiceRegistryLookupProperties.Template = 'template1';` This setting is valid only for the RegistryLookup node, and is ignored by the EndpointLookup node.
MatchPolicy	This setting overrides the Match Policy property on the node; for example: `SET OutputLocalEnvironment.ServiceRegistryLookupProperties.MatchPolicy = 'One';` Valid values are One and All.
DepthPolicy	This setting overrides the Depth Policy property on the RegistryLookupnode; for example: `SET OutputLocalEnvironment.ServiceRegistryLookupProperties.DepthPolicy = 'MatchOnly';` Valid values are: • MatchOnly for Return matched only (Depth = 0) • MatchShowRel for Return matched showing immediate relationships (For compatibility only) • MatchPlusImmediate for Return matched plus immediate related entities (Depth = 1) • MatchPlusAll for Return matched plus all related entities (Depth = -1) The MatchShowRel property provides compatibility with versions of WebSphere Message Broker before Version 6.1.0.4, by using the output format that was used in those previous versions. This option is deprecated, and should not be used if you are creating a new message flow. Consider migrating existing message flows to use one of the other options.

UserProperties	This setting overrides the User Properties property on the node. You can specify more than one user-defined property in the local environment; for example:
	`SET OutputLocalEnvironment.ServiceRegistryLookupProperties.UserProperties.property1 = 'value1';`
	`SET OutputLocalEnvironment.ServiceRegistryLookupProperties.UserProperties.property2 = 'value2';`
	
	You can remove a user-defined property from the local environment by setting its value to NULL; for example:
	`SET OutputLocalEnvironment.ServiceRegistryLookupProperties.UserProperties.property1 = NULL;`
	
	You can use the node properties editor at design time to specify ESQL paths or XPath expressions to read the value for a user property at run time from a field in the message tree. However, the override values that you set in the local environment are the string values that are used in the query.

Classification	This setting overrides the Classification property on the node; for example:
	`SET OutputLocalEnvironment.ServiceRegistryLookupProperties.Classification = 'http://www.ibm.com/xmlns/prod/serviceregistry/6/0/governance/DefaultLifecycle#InitialState0';`
	
	You can specify more than one classification in the local environment. For example:
	`SET OutputLocalEnvironment.ServiceRegistryLookupProperties.Classification[1] = 'http://www.ibm.com/xmlns/prod/serviceregistry/6/0/governance/DefaultLifecycle#InitialState0';`
	`SET OutputLocalEnvironment.ServiceRegistryLookupProperties.Classification[2] = 'http://www.ibm.com.policy/GovernancePolicyDomain';`

EndpointLookup node output

Use an EndpointLookup node to retrieve the endpoint addresses for WSDL service definitions held in WebSphere Service Registry and Repository (WSRR).

The input message is not changed by the EndpointLookup node. Instead, the local environment is updated to contain details of the endpoints retrieved by the query specified by the node and any local environment overrides.

You can configure the EndpointLookup node to dynamically set the service endpoint address for services that will be invoked by a subsequent SOAP or HTTP request node. The EndpointLookup node sets the destination URL in the local environment overrides for those nodes. See the following sample for an example of how to do this:

- WSRR Connectivity

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

EndpointLookup node output if the Match Policy property is set to One

If the Match Policy property of the node is set to One, theEndpointLookup node inserts the endpoint URL retrieved by the query into the local environment in an ITService entry in the local environment under ServiceRegistry, and sets the destination overrides for the SOAP and HTTP request nodes that can be connected directly to its output terminal. The following locations are updated:

- LocalEnvironment.Destination.SOAP.Request.Transport.HTTP.WebServiceURL
- LocalEnvironment.Destination.HTTP.RequestURL

These settings override the Web service URL property of the SOAPRequest, SOAPAsyncRequest, and HTTPRequest nodes, allowing a dynamic call to a Web service provider.

The following example shows typical output from the EndpointLookup node when the Match Policy is set to One. (Other entries might exist in the local environment depending on previous processing in the flow.)

```
<LocalEnvironment>
  <Destination>
    <SOAP>
      <Request>
        <Transport>
          <HTTP>
            <WebServiceURL>http://localhost:9081/DemoCustomerWeb/
              services/DemoCustomer</WebServiceURL>
          </HTTP>
        </Transport>
      </Request>
    </SOAP>
    <HTTP>
      <RequestURL>http://localhost:9081/DemoCustomerWeb/
        services/DemoCustomer
    </RequestURL>
   </HTTP>
  </Destination>
  <ServiceRegistry>
    <ITService>
      <Endpoint>
        <Address>http://localhost:9081/DemoCustomerWeb/
          services/DemoCustomer</Address>
```

```
        <PortType>
          <name>DemoCustomer</name>
          <namespace>http://demo.sr.eis.ibm.com</namespace>
          <version>1.0</version>
        </PortType>
        <Property>
          <name>policy</name>
          <value>RM</value>
        </Property>
        <Property>
          <name>country</name>
          <value>China</value>
        </Property>
        <Classification>http://eis.ibm.com/ServiceRegistry/
          GenericObjecttypes#Routing</Classification>
      </Endpoint>
    </ITService>
  </ServiceRegistry>
</LocalEnvironment>
```

EndpointLookup node output if the Match Policy property is set to All

If the Match Policy is set to All the EndpointLookup node writes an ITService entry in the local environment location ServiceRegistry for each endpoint retrieved by the query.

The following example shows typical output from the EndpointLookup node when the Match Policy is set to All. (Other entries might exist in the local environment depending on previous processing in the flow.)

```
<LocalEnvironment>
  <ServiceRegistry>
    <ITService>
      <Endpoint>
        <Address>http://localhost:9081/DemoCustomerWeb/
          services/DemoCustomer</Address>
        <PortType>
          <name>DemoCustomer</name>
          <namespace>http://demo.sr.eis.ibm.com</namespace>
          <version>1.0</version>
        </PortType>
        <Property>
          <name>policy</name>
          <value>RM</value>
        </Property>
        <Property>
          <name>country</name>
          <value>China</value>
        </Property>
        <Classification>http://eis.ibm.com/ServiceRegistry/
          GenericObjecttypes#Routing</Classification>
      </Endpoint>
    </ITService>
    <ITService>
      <Endpoint>
        <Address>http://localhost:9081/DemoCustomerWeb/
          services/DemoCustomer2</Address>
        <PortType>
          <name>DemoCustomer2</name>
          <namespace>http://demo.sr.eis.ibm.com</namespace>
          <version>1.0</version>
        </PortType>
```

```
      </Endpoint>
    </ITService>
  </ServiceRegistry>
</LocalEnvironment>
```

RegistryLookup node output

Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. The input message is not modified.

This topic contains the following sections:
- "Setting the Depth Policy property"
- "Local environment output tree" on page 796
- "Migrating a message flow that uses the MatchShowRel option" on page 798
- "Examples" on page 798

Setting the Depth Policy property

The Depth Policy property on the RegistryLookup node specifies how much data is returned for each matched entity. The following table shows the valid values.

Depth Policy property value	Local environment override value	Data returned
Return matched only (Depth = 0)	MatchOnly	Just the matched entities
Return matched showing immediate relationships (For compatibility only)	MatchShowRel	The matched entities and additional references
Return matched plus immediate related entities (Depth = 1)	MatchPlusImmediate	The matched entities and the immediate related child entities
Return matched plus all related entities (Depth = -1)	MatchPlusAll	The matched entities and all the related child entities

Use MatchShowRel for compatibility with versions of WebSphere Message Broker before Version 6.1.0.4. The local environment output for MatchShowRel is shown in Example 1 and follows the format that was used in those previous versions. However, the MatchShowRel option is deprecated, and provided only for compatibility with previous versions. Do not use MatchShowRel if you are creating a new message flow, and consider migrating existing message flows to use one of the other options.

Use the MatchOnly option to retrieve just the individual entities matched by the search criteria. This option is efficient, however the local environment output does not contain any information about entities related to the matched entities.

Use the MatchPlusImmediate option to retrieve the entities matched by the search criteria, and the related child entities. This option offers a useful compromise, allowing you to access the immediate relations of the matched entities, while still restricting the total amount of data retrieved.

Use one of the MatchOnly or MatchPlusImmediate options when migrating a message flow that uses the deprecated MatchShowRel option. If your original

message flow used the relationship information in the matched entities, use the MatchPlusImmediate option. See “Migrating a message flow that uses the MatchShowRel option” on page 798.

Use the MatchPlusAll option to retrieve the entities matched by the search criteria, and all the related child entities. Use this option only if your message flow needs access to more than the immediate relations of the matched entities, because it retrieves considerably more data than the MatchPlusImmediate option, see “Local environment output tree.”

Local environment output tree

The local environment output tree has a different format when the deprecated MatchShowRel option of the Depth Policy property is used. The following table describes the differences in the local environment output tree format.

MatchOnly, MatchPlusImmediate, and MatchPlusAll options	**MatchShowRel option**
The ServiceRegistry folder element is owned by the XMLNSC compiler.	No owning parser on the ServiceRegistry folder element, and each Entity element is owned by the XMLNS parser
The ServiceRegistry tree does not use unnecessary namespaces.	Namespaces are attached to all UserDefined folder elements, meaning that the path specified must declare and use the relevant namespace to access fields within these folders.
The ServiceRegistry tree is optimized through use of the XMLNSC parser.	The output tree contains a number of XML declaration, pcdata, and white space elements that have no business significance.
In WSRR, binary data is represented as a GenericDocument with a `content` attribute. The `content` attribute is represented in the local environment as XMLNSC type Attribute+base64Binary, (0x03000160). You can access the unencoded binary data directly as `Entity.content`. However, because of the special XMLNSC element type, the data is automatically base64 encoded if the tree is serialized.	Binary content is represented as a base64 encoded character string in the `content` attribute. The message flow must invoke a Java method to decode the value if the original binary data is required.
The retrieved entities are not modified to add any user properties.	The matched Entities appear in the local environment with a user property called `WSRREncoding`. This property has no specific significance to message flow processing. If there is a user property called `WSRREncoding` defined for the entity, the defined value is used, otherwise the `WSRREncoding` property is added with `value="DEFAULT"`.

The MatchPlusImmediate and MatchPlusAll options result in a graph of entities being returned when WSRR executes the query. Each entity in the graph can refer to other entities. There are two types of relationship that cannot be expressed directly in a hierarchical tree:

- Cyclic references - a graph can contain an entity from which you can follow relationships that arrive back at the same entity.

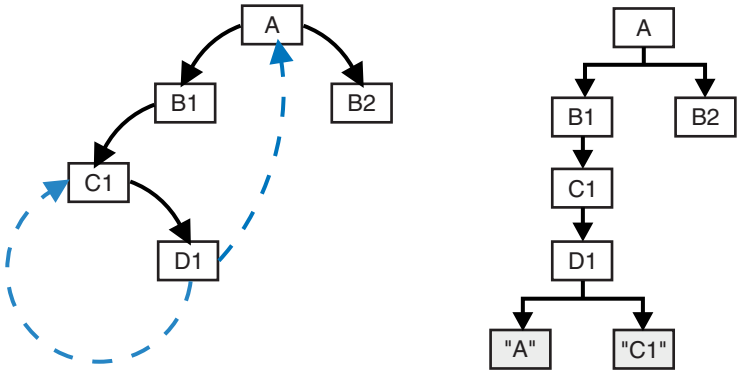
- Multiple references - a graph can contain an entity that is referenced by multiple entities.

The representation of these relationships in the local environment output tree is described in the following text.

The WSRR graph is represented in the local environment output tree as follows:

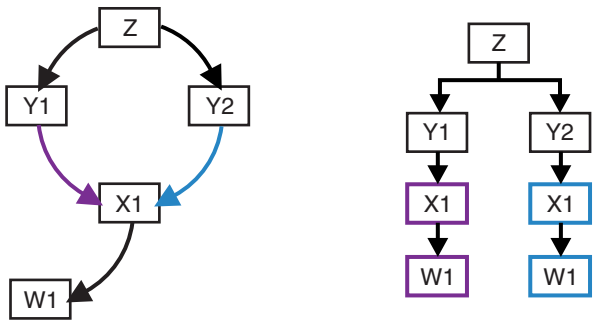
- The matched entities that are returned in the WSRR graph are represented as Entity elements as the first children of the LocalEnvironment.ServiceRegistry. The properties of the matched entities are represented as child elements.
- A matched entity can contain references to other entities. If MatchPlusImmediate is set, these references are represented as Entity child elements of the matched entity. If MatchPlusAll is set, the same rule is applied recursively to these children.
- Cyclic references - if an entity references another entity that is one of its ancestors in the WSRR graph, the referenced entity is represented as an EntityRef element. An EntityRef element does not represent the target entity directly, but provides information to identify it. This prevents the possibility of entering cyclic loops when navigating the tree.

  The following diagram shows references from entity D1 in the graph to entities C1 and A that are ancestral; therefore, Entity D1 in the message tree contains EntityRef elements for A and C1.



- Multiple references - if an entity in the WSRR SDO graph is referenced by more than one other entity, it is represented as a separate Entity element in the message tree each time it is referenced. The Entity element is cloned, along with any entities that it refers to.

  The following diagram shows that entity X1 in the graph is referenced by entities Y1 and Y2, so Entity X1 and Entity W1 that it references are modeled twice in the message tree.



Migrating a message flow that uses the MatchShowRel option

When you migrate an existing message flow that uses the deprecated MatchShowRel option of the Depth Policy property, the local environment output tree will have a different format. The table shown earlier describes the differences in the local environment output tree formats, and indicates what you must modify in the message flow to access the data in the updated format; see "Local environment output tree" on page 796.

Examples

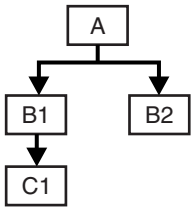
The following examples show typical output from the RegistryLookup node:

- Example 1 shows the full RegistryLookup node output in two cases for a query that returns two versions of a concept entity. In both cases the Match Policy property is set to All. In the first case the Depth Policy property is set to Return matched plus immediate related entities (Depth = 1) , and in the second case the Depth Policy property is set to Return matched only, showing immediate relationships (For compatibility only). This example also shows example ESQL to read elements of the output. See "RegistryLookup node output: example 1"
- Example 2 shows the structure of RegistryLookup node output for all possible values of the Depth Policy property for a query on a concept entity that has a number of user relationships to other concept entities. See "RegistryLookup node output: example 2" on page 802
- Example 3 shows the structure of RegistryLookup node output for a query on an entity having metadata relationships and user-defined relationships, using a Depth Policy property value of Return matched plus all related entities (Depth = -1) . See "RegistryLookup node output: example 3" on page 805.

RegistryLookup node output: example 1

Example showing the full RegistryLookup node output in two cases for a query that returns two versions of a concept entity. In both cases the Match Policy property is set to All. In the first case the Depth Policy property is set to Return matched plus immediate related entities (Depth = 1) , and in the second case the Depth Policy property is set to Return matched only, showing immediate relationships (For compatibility only). This example also shows example ESQL to read elements of the output.

This example shows the ServiceRegistry message tree that is stored in the LocalEnvironment when an entity called ConceptA1 is retrieved from WebSphere Service Registry and Repository (WSRR). ConceptA1 is defined in WSRR with 2 versions: 1.0 which is deprecated, and 2.0 which is classified as initial state. Additional properties and relationships have been added to the 2.0 version. The following diagram shows the relationships between the elements in the message tree.



The following message trees show the different values of the Depth Policy property.

- Depth Policy property set to Return matched plus immediate related entities (Depth = 1)

  The following ESQL was used to create a serialization of the output of the ServiceRegistry node:

```
SET OutputRoot.XMLNSC.Result.ServiceRegistry =
  InputLocalEnvironment.ServiceRegistry;
```

  The ServiceRegistry folder element is owned by the XMLNSC parser so you can invoke a like parser tree copy to OutputRoot.XMLNSC. The following XML, which has been formatted, is produced when writing this output root tree.

```
<ServiceRegistry>
  <Entity
    bsrURI="33a9ad33-d4a4-442e.b3a6.37e62137a605"
    name="ConceptA1"
    namespace="http://www.examples.com/ConceptA1"
    version="2.0"
    description="A version 2 of the existing one"
    owner="UNAUTHENTICATED"
    lastModified="1230116323343"
    creationTimestamp="1227176757406"
    lastModifiedBy="UNAUTHENTICATED"
    primaryType="">
    <classificationURIs>
      http://www.ibm.com/xmlns/prod/serviceregistry/6/0/governance/
        DefaultLifecycle#InitialState0
    </classificationURIs>
    <classificationURIs>
      http://www.ibm.com.policy/GovernancePolicyDomain
    </classificationURIs>
    <userDefinedProperties name="property1" value="value1 for v1" />
    <userDefinedProperties name="property2" value="value1 for v2" />
    <userDefinedProperties name="property3" value="value1 for v3" />
    <userDefinedRelationships name="ContainsChildren">
      <targetEntities>
        <Entity
          bsrURI="8203cb82-8827-4757.99e1.36de6036e1af"
          name="ConceptB1"
          namespace="http://www.examples.com/ConceptB1"
          version="2.0"
          description="Next revision of this concept"
          owner="UNAUTHENTICATED"
          lastModified="1227191748250"
          creationTimestamp="1227177460156"
          lastModifiedBy="UNAUTHENTICATED"
          primaryType="" />
        <Entity
          bsrURI="a0d2bba0-f395-45bc.929e.04d143049eb5"
          name="ConceptB2"
          namespace="http://www.examples.com/ConceptB2" version="1.0"
          description="Testing"
          owner="UNAUTHENTICATED"
          lastModified="1227191700812"
          creationTimestamp="1227177334515"
          lastModifiedBy="UNAUTHENTICATED"
          primaryType="" />
      </targetEntities>
    </userDefinedRelationships>
    <userDefinedRelationships name="ReferTo">
      <targetEntities>
        <Entity
          bsrURI="81e45381-a9be-4ea4.9519.53657953196d"
          name="ConceptC1"
          namespace="http://www.examples.com/ConceptC1"
          version="1.0"
          description="Test stuff C1"
```

```
          owner="UNAUTHENTICATED"
          lastModified="1227874855140"
          creationTimestamp="1227177519609"
          lastModifiedBy="UNAUTHENTICATED" primaryType="" />
      </targetEntities>
    </userDefinedRelationships>
  </Entity>
  <Entity
    bsrURI="b68952b6-8d68-4840.8e5e.a3716da35e2e"
    name="ConceptA1"
    namespace="http://www.examples.com/ConceptA1"
    version="1.0"
    description="The original concept"
    owner="UNAUTHENTICATED"
    lastModified="1229030287593"
    creationTimestamp="1227173773250"
    lastModifiedBy="UNAUTHENTICATED"
    primaryType="">
    <classificationURIs>
      http://www.ibm.com/xmlns/prod/serviceregistry/6/0/governance/
        DefaultLifecycle#Deprecate
    </classificationURIs>
    <userDefinedProperties name="property1" value="value1" />
    <userDefinedProperties name="property2" value="value2" />
    <userDefinedRelationships name="ContainsChildren">
      <targetEntities>
        <Entity
          bsrURI="7d5fd37d-e90a-4ab0.89eb.d25b81d2ebec"
          name="ConceptB1"
          namespace="http://www.examples.com/ConceptB1" version="1.0"
          description="" owner="UNAUTHENTICATED"
          lastModified="1227874785062"
          creationTimestamp="1227177401265"
          lastModifiedBy="UNAUTHENTICATED"
          primaryType="" />
        <Entity
          bsrURI="a0d2bba0-f395-45bc.929e.04d143049eb5"
          name="ConceptB2"
          namespace="http://www.examples.com/ConceptB2" version="1.0"
          description="Testing" owner="UNAUTHENTICATED"
          lastModified="1227191700812"
          creationTimestamp="1227177334515"
          lastModifiedBy="UNAUTHENTICATED"
          primaryType="" />
      </targetEntities>
    </userDefinedRelationships>
  </Entity>
</ServiceRegistry>
```

The following ESQL shows how to retrieve the values from the ServiceRegistry message tree in the LocalEnvironment when the Depth Policy property is set to Return matched plus immediate related entities (Depth = 1).

```
DECLARE c1 CHARACTER;

-- Following sets c1 to "ConceptA1" by indexing the first entity
SET c1 = InputLocalEnvironment.ServiceRegistry.Entity[1].name;
-- Following sets c1 to "2.0" by indexing the first entity
SET c1 = InputLocalEnvironment.ServiceRegistry.Entity[1].version;
-- Following sets c1 to "property1" by indexing the first
--   userDefinedProperties within the first entity
SET c1 = InputLocalEnvironment.ServiceRegistry.Entity[1].
  userDefinedProperties[1].name;
-- Following sets c1 to "value1 for v2" by indexing the first
--   userDefinedProperties within the first entity
SET c1 = InputLocalEnvironment.ServiceRegistry.Entity[1].
  userDefinedProperties[1].value;
```

- Depth Policy property set to Return matched showing immediate relationships (For compatibility only)

  The following ESQL was used to create a serialization of the output of the ServiceRegistry node:

```
DECLARE I INTEGER 1;
DECLARE J INTEGER;
SET J = CARDINALITY(InputLocalEnvironment.ServiceRegistry.Entity[]);
WHILE I < J DO
  SET OutputRoot.XMLNS.ServiceRegistry.Entity[I] =
    InputLocalEnvironment.ServiceRegistry.Entity[I];
  SET I = I + 1;
END WHILE;
```

  The ServiceRegistry folder does not have an owning parser so you must navigate to the ""elements and initiate a like-parser-copy to the XMLNS owned output root tree. The following XML is produced when writing this output root tree.

```
<ServiceRegistry>
  <Entity
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:sdo="http://www.ibm.com/xmlns/prod/serviceregistry/6/0/sdo"
    xsi:type="sdo:GenericObject"
    bsrURI="33a9ad33-d4a4-442e.b3a6.37e62137a605"
    name="ConceptA1"
    namespace="http://www.examples.com/ConceptA1"
    version="2.0"
    description="A version 2 of the existing one"
    owner="UNAUTHENTICATED"
    lastModified="1229439847694"
    creationTimestamp="1227176757406"
    lastModifiedBy="UNAUTHENTICATED"
    primaryType="">
    <sdo:classificationURIs>
      http://www.ibm.com/xmlns/prod/serviceregistry/6/0/governance/
        DefaultLifecycle#InitialState0
    </sdo:classificationURIs>
    <sdo:classificationURIs>
      http://www.ibm.com.policy/GovernancePolicyDomain
    </sdo:classificationURIs>
    <sdo:userDefinedRelationships
      name="ContainsChildren"
      targets="8203cb82-8827-4757.99e1.36de6036e1af
        a0d2bba0-f395-45bc.929e.04d143049eb5" />
    <sdo:userDefinedRelationships
      name="ReferTo"
      targets="81e45381-a9be-4ea4.9519.53657953196d" />
    <sdo:userDefinedProperties name="property1" value="value1 for v2" />
    <sdo:userDefinedProperties name="property2" value="value2 for v2" />
    <sdo:userDefinedProperties name="property3" value="value3 for v2" />
    <sdo:userDefinedProperties name="WSRRencoding" value="DEFAULT" />
  </Entity>
  <Entity>
    <NS1:type
      xmlns:NS1="http://www.w3.org/2001/XMLSchema-instance">
      sdo:GenericObject
    </NS1:type>
    <bsrURI>b68952b6-8d68-4840.8e5e.a3716da35e2e</bsrURI>
    <name>ConceptA1</name>
    <namespace>http://www.examples.com/ConceptA1</namespace>
    <version>1.0</version>
    <description>The original concept</description>
    <owner>UNAUTHENTICATED</owner>
    <lastModified>1229030287593</lastModified>
    <creationTimestamp>1227173773250</creationTimestamp>
```

```
    <lastModifiedBy>UNAUTHENTICATED</lastModifiedBy>
    <primaryType></primaryType>
    <NS2:classificationURIs
      xmlns:NS2="http://www.ibm.com/xmlns/prod/serviceregistry/6/0/sdo">
      http://www.ibm.com/xmlns/prod/serviceregistry/6/0/governance/
        DefaultLifecycle#Deprecate
    </NS2:classificationURIs>
    <NS3:userDefinedRelationships
      xmlns:NS3="http://www.ibm.com/xmlns/prod/serviceregistry/6/0/sdo">
    </NS3:userDefinedRelationships>
    <NS4:userDefinedProperties
      xmlns:NS4="http://www.ibm.com/xmlns/prod/serviceregistry/6/0/sdo">
    </NS4:userDefinedProperties>
    <NS5:userDefinedProperties
      xmlns:NS5="http://www.ibm.com/xmlns/prod/serviceregistry/6/0/sdo">
    </NS5:userDefinedProperties>
    <NS6:userDefinedProperties
      xmlns:NS6="http://www.ibm.com/xmlns/prod/serviceregistry/6/0/sdo">
    </NS6:userDefinedProperties>
  </Entity>
</ServiceRegistry>
```

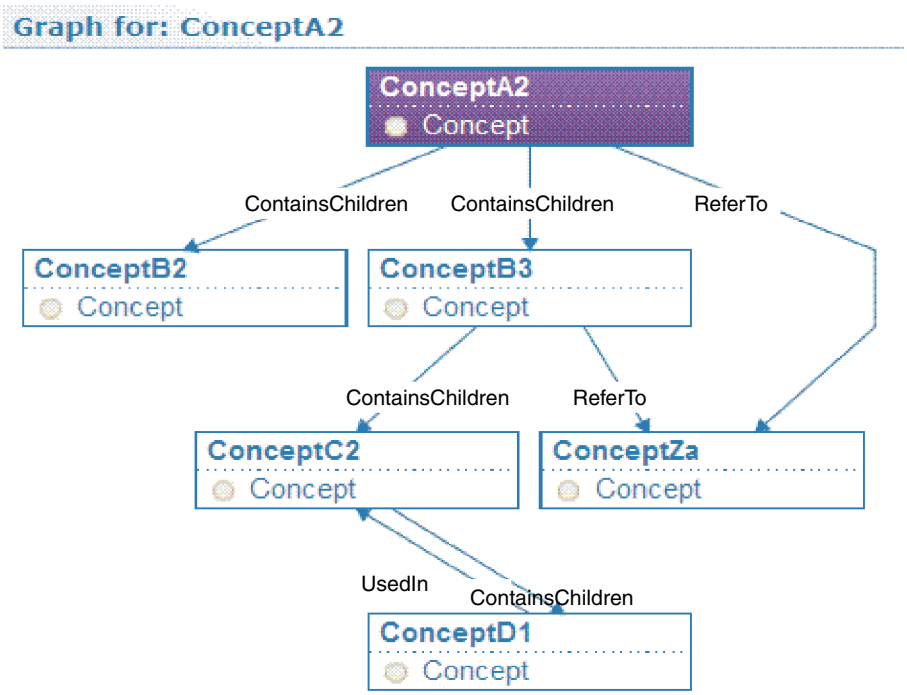
The following ESQL shows how to retrieve the values from the ServiceRegistry message tree in the LocalEnvironment when the Depth Policy property is set to Return matched showing immediate relationships (For compatibility only). Note that in this case it is necessary to use the namespace qualifier.

```
DECLARE ns1 NAMESPACE 'http://www.ibm.com/xmlns/prod/serviceregistry/6/0/sdo';
DECLARE c1 CHARACTER;

-- Following sets c1 to "ConceptA1" by indexing the first entity
SET c1 = InputLocalEnvironment.ServiceRegistry.Entity[1].name;
-- Following sets c1 to "2.0" by indexing the first entity
SET c1 = InputLocalEnvironment.ServiceRegistry.Entity[1].version;
-- Following sets c1 to "property1" by indexing the first
--   userDefinedProperties within the first entity
SET c1 = InputLocalEnvironment.ServiceRegistry.Entity[1].ns1:
userDefinedProperties[1].name;
-- Following sets c1 to "value1 for v2" by indexing the first
--   userDefinedProperties within the first entity
SET c1 = InputLocalEnvironment.ServiceRegistry.Entity[1].ns1:
userDefinedProperties[1].value;
```

RegistryLookup node output: example 2

Example showing the structure of RegistryLookup node output for all possible values of the Depth Policy property for a query on a concept entity that has a number of user relationships to other concept entities.

This example shows the ServiceRegistry message trees that are stored in the LocalEnvironment when the Concepts shown in the following WebSphere Service Registry and Repository graph are retrieved. The graph has been annotated with the relationship names to clarify the elements in the message tree.



The following ServiceRegistry message trees have some elements replaced by ... to emphasis the structure of the tree. Likewise, the bsrURIs have been truncated.

The following shows the message trees for each possible value of the Depth Policy property:

- Return matched only (Depth = 0)

```
ServiceRegistry
  Entity
    type   = sdo:GenericObject
    bsrURI = a2e62137a605
    name   = ConceptA2
    ...
```

- Return matched showing immediate relationships (For compatibility only). The entities contain elements showing the details of relationships, but only provide a list of the bsrURIs for the related child entities.

  This value of the Depth Policy property is deprecated, so you should use of the other options. The output tree structure produced when using this value is not compatible with those from the other values for the Depth Policy property. In particular, note the namespace qualifications.

```
ServiceRegistry
  Entity
    type   = sdo:GenericObject
    bsrURI = a2e62137a605
    name   = ConceptA2
    ...
    ns1:userDefinedRelationships
      name    = ContainsChildren
      targets = b2f73637f6e8 b3de6036e1af
    ns1:userDefinedRelationships
      name    = ReferTo
      targets = zac084d6b804
```

- Return matched plus immediate related entities (Depth = 1). The entities contain elements showing the details of relationships, and the details of the related child entities.

```
ServiceRegistry
  Entity
    type   = GenericObject
    bsrURI = a2e62137a605
    name   = ConceptA2
    ...
    userDefinedRelationships
      name = ContainsChildren
      targetEntities
        Entity
          bsrURI = b2f73637f6e8
          name   = ConceptB2
          ...
        Entity
          bsrURI = b3de6036e1af
          name   = ConceptB3
          ...
          userDefinedRelationships
            name    = ContainsChildren
            targets = c26e43ac45a
          userDefinedRelationships
            name    = ReferTo
            targets = zac084d6b804
    userDefinedRelationships
      name = ReferTo
      targetEntities
        Entity
          bsrURI = zac084d6b804
          name   = ConceptZa
          ...
```

- Return matched plus all related entities (Depth = -1). The entities contain elements showing the details of relationships, and the details of the all related child entities. ConceptD1 uses an EntityRef element to refer to its ancestor ConceptC2. ConceptZa appears twice in the tree as it is referenced by both ConceptA2 and ConceptB3.

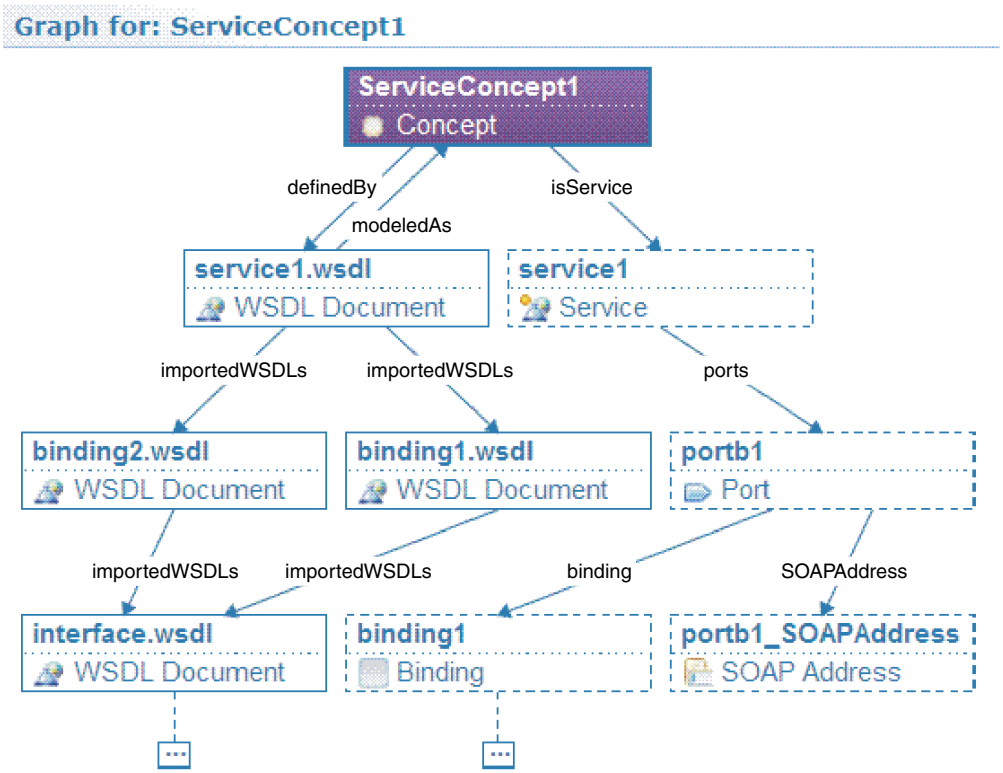
```
ServiceRegistry
  Entity
    type   = sdo:GenericObject
    bsrURI = a2e62137a605
    name   = ConceptA2
    ...
    userDefinedRelationships
      name = ContainsChildren
      targetEntities
        Entity
          bsrURI = b2f73637f6e8
          name   = ConceptB2
          ...
        Entity
          bsrURI = b3de6036e1af
          name   = ConceptB3
          ...
          userDefinedRelationships
            name = ContainsChildren
            targetEntities
              Entity
                bsrURI = c26e43ac45a
                name   = ConceptC2
                ...
                userDefinedRelationships
                  name = ContainsChildren
                  targetEntities
```

```
                Entity
                  bsrURI = d16e43ac763
                  name   = ConceptD1
                  ...
                  userDefinedRelationships
                    name = UsedIn
                    targetEntities
                      EntityRef
                        bsrURI = c26e43ac45a
                        name   = ConceptC2
      userDefinedRelationships
        name = ReferTo
        targetEntities
          Entity
            bsrURI = zac084d6b804
            name   = ConceptZa
            ...
userDefinedRelationships
  name = ReferTo
  targetEntities
    Entity
      bsrURI = zac084d6b804
      name   = ConceptZa
      ...
```

RegistryLookup node output: example 3

Example showing the structure of RegistryLookup node output for a query on an entity having metadata relationships and user-defined relationships, using a Depth Policy property value of Return matched plus all related entities (Depth = -1) .

This example shows the ServiceRegistry message tree that is stored in the LocalEnvironment when the relationships shown in the following WebSphere Service Registry and Repository graph are retrieved using a Depth Policy value of Return matched plus all related entities (Depth = -1). The graph has been annotated with the relationship names to clarify the elements in the message tree.



The following ServiceRegistry message tree has some elements replaced by ... to emphasis the structure of the tree.

```
ServiceRegistry
  Entity
    name = ServiceConcept1
    ...
    userDefinedRelationships
      name = modeledAs
      targetEntities
        Entity
          name = service1.wsdl
          ...
          userDefinedRelationships
            name = definedBy
            targetEntities
              EntityRef
                bsrURI = c26e43ac45a
                name = ServiceConcept1
          metaRelationships
            name = importedWSDLs
            targetEntities
              Entity
                name = binding1.wsdl
                ...
              Entity
                name = binding2.wsdl
                ...
    userDefinedRelationships
      name = isService
      targetEntities
        Entity
          name = service1
          ...
          metaRelationships
```

```
name = ports
targetEntities
  Entity
    name = portb1
    ...
    metaRelationships
      name = binding
      targetEntities
        Entity
          name = binding1
          ...
    metaRelationships
      name = SOAPAddress
      targetEntities
        Entity
          name = portb1_SOAPAddress
          ...
```

External standards

WebSphere Message Broker support for Web services conforms to a number of industry standards and specifications.

This section contains the topics that describe the WebSphere Message Broker support for Web services.

SOAP 1.1 and 1.2

SOAP is a lightweight, XML-based, protocol for exchange of information in a decentralized, distributed environment.

The protocol consists of three parts:

- An envelope that defines a framework for describing what is in a message and how to process it.
- A set of encoding rules for expressing instances of application-defined data types.
- A convention for representing remote procedure calls and responses.

SOAP can be used with other protocols, such as HTTP.

The specifications for SOAP are published by the World Wide Web Consortium (W3C).

- World Wide Web Consortium (W3C)

The specification for SOAP 1.1 is described in:

- Simple Object Access Protocol 1.1

This specification has not been endorsed by the W3C, but forms the basis for the SOAP 1.2 specification. The specification for SOAP 1.1 expands the SOAP acronym to Simple Object Access Protocol.

SOAP 1.2 is a W3C recommendation and is published in two parts:

- Part 1: Messaging Framework.
- Part 2: Adjuncts.

The specification also includes a primer that is intended to provide a tutorial on the features of the SOAP Version 1.2 specification, including usage scenarios. The specification for SOAP 1.2 does not expand the acronym. The primer is published at:

- SOAP 1.2 Primer

SOAP with Attachments

SOAP with Attachments (SwA) allows you to use SOAP 1.1 or SOAP 1.2 messages wrapped by MIME.

The SOAP with Attachments (SwA) specification is published as a formal submission by the World Wide Web Consortium (W3C):

- World Wide Web Consortium (W3C)

SwA uses the following specifications, described at:

- http://www.w3.org/TR/2000/NOTE-SOAP-attachments-20001211
- http://www.ws-i.org/Profiles/AttachmentsProfile-1.0.html

SOAP MTOM

SOAP Message Transmission Optimization Mechanism (MTOM) is one of a related pair of specifications that define conceptually how to optimize the transmission and format of a SOAP message.

MTOM defines:

- How to optimize the transmission of base64binary data in SOAP messages in abstract terms
- How to implement optimized MIME multipart serialization of SOAP messages in a binding independent way using XOP

The implementation of MTOM relies on the related XML-binary Optimized Packaging (XOP) specification. Because these two specifications are so closely linked, they are normally referred to as MTOM/XOP.

The specification is published by the World Wide Web Consortium (W3C) as a W3C Recommendation at SOAP Message Transmission Optimization Mechanism. For further information refer to the following links:

- World Wide Web Consortium (W3C)

- SOAP Message Transmission Optimization Mechanism

WSDL Version 1.1

Web Services Description Language (WSDL) is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information.

The operations and messages are described abstractly, then bound to a concrete network protocol and message format to define an endpoint. Related concrete end points are combined into abstract endpoints (services).

WSDL is extensible to allow the description of endpoints and their messages regardless of what message formats or network protocols are used to communicate. The WSDL 1.1 specification only defines bindings that describe how to use WSDL in conjunction with:

- SOAP 1.1
- HTTP GET
- HTTP POST
- MIME

The specification for WSDL 1.1 is published by the World Wide Web Consortium (W3C) as a W3C Note at WSDL Version 1.1.

- World Wide Web Consortium (W3C)
- WSDL Version 1.1

WS-I Simple SOAP Binding Profile Version 1.0

WS-I Simple SOAP Binding Profile Version 1.0 (SSBP 1.0) is a set of non-proprietary Web services specifications, along with clarifications and amendments to those specifications which promote interoperability.

The SSBP 1.0 is derived from the WS-I Basic Profile 1.0 requirements that relate to the serialization of the envelope and its representation in the message.

WS-I Basic Profile 1.0 is split into two separately published profiles. These profiles are:

- WS-I Basic Profile Version 1.1
- WS-I Simple SOAP Binding Profile Version 1.0

Together these two profiles supersede the WS-I Basic Profile Version 1.0.

The specification for SSBP 1.0 is published by the Web Services Interoperability Organization (WS-I):

- Web Services Interoperability Organization (WS-I)

The specification for SSBP 1.0 can be found at:

- WS-I Simple SOAP Binding Profile Version 1.0

WS-I Basic Profile Version 1.1

WS-I Basic Profile Version 1.1 (WS-I BP 1.1) is a set of non-proprietary Web services specifications, along with clarifications and amendments to those specifications, which together promote interoperability between different implementations of Web services.

The WS-I BP 1.1 is derived from Basic Profile Version 1.0 by incorporating its published errata and separating out the requirements that relate to the serialization of envelopes and their representation in messages. These requirements are now part of the Simple SOAP Binding Profile Version 1.0.

To summarize, the WS-I Basic Profile Version 1.0 is split into two separately published profiles. These profiles are:

- WS-I Basic Profile Version 1.1
- WS-I Simple SOAP Binding Profile Version 1.0

Together these two profiles supersede the WS-I Basic Profile Version 1.0.

The reason for this separation is to enable the Basic Profile 1.1 to be composed with any profile that specifies envelope serialization, including the Simple SOAP Binding Profile 1.0.

The specification for WS-I BP 1.1 is published by the Web Services Interoperability Organization (WS-I):

- Web Services Interoperability Organization (WS-I)

The specification for WS-I BP 1.1 can be found at:

- WS-I Basic Profile Version 1.1

WSDL 1.1 Binding Extension for SOAP 1.2

WSDL 1.1 Binding Extension for SOAP 1.2 is a specification that defines the binding extensions that are required to indicate that Web service messages are bound to the SOAP 1.2 protocol.

The aim of this specification is to provide functions that are comparable with the binding for SOAP 1.1.

This specification is published as a formal submission request by the World Wide Web Consortium (W3C):

- World Wide Web Consortium (W3C)

The WSDL 1.1 Binding Extension for SOAP 1.2 specification is described at:

- http://www.w3.org/Submission/wsdl11soap12/

XML-Binary Optimised Packaging (XOP)

XML-binary Optimized Packaging (XOP) is one of a related pair of specifications that define how to efficiently serialize XML Infosets that have certain types of content.

XOP defines how to efficiently serialize XML Infosets that have certain types of content by:

- Packaging the XML in some format. This is called the XOP package. The specification mentions MIME Multipart/Related but does not limit it to this format.
- Re-encoding all or part of base64binary content to reduce its size.
- Placing the base64binary content elsewhere in the package and replacing the encoded content with XML that references it.

XOP is used as an implementation of the MTOM specification, which defines the optimization of SOAP messages. Because these two specifications are so closely linked, they are normally referred to as MTOM/XOP.

The specification is published by the World Wide Web Consortium (W3C) as a W3C Recommendation XML-binary Optimized Packaging (XOP):

- World Wide Web Consortium (W3C)
- XML-binary Optimized Packaging (XOP)

SOAP Binding for MTOM 1.0

SOAP 1.1 Binding for MTOM 1.0 is a specification that describes how to use the SOAP Message Transmission Optimization Mechanism (MTOM) and XML-binary Optimized Packaging (XOP) specifications with SOAP 1.1.

This specification defines the minimum changes required to enable MTOM and XOP to be used interoperably with SOAP 1.1 and to reuse the SOAP 1.2 MTOM/XOP implementation.

The SOAP 1.1 Binding for MTOM 1.0 specification is published as a formal submission by the World Wide Web Consortium (W3C):

- World Wide Web Consortium (W3C)

The SOAP 1.1 Binding for MTOM 1.0 specification is described at:

- http://www.w3.org/Submission/soap11mtom10/

Web Services Security: SOAP Message Security

Web Services Security (WSS): SOAP Message Security is a set of enhancements to SOAP messaging that provides message integrity and confidentiality. WSS: SOAP Message Security is extensible, and can accommodate a variety of security models and encryption technologies.

WSS: SOAP Message Security provides three main mechanisms that can be used independently or together:

- The ability to send security tokens as part of a message, and for associating the security tokens with message content
- The ability to protect the contents of a message from unauthorized and undetected modification (message integrity)
- The ability to protect the contents of a message from unauthorized disclosure (message confidentiality).

WSS: SOAP Message Security can be used in conjunction with other Web service extensions and application-specific protocols to satisfy a variety of security requirements.

The specification is published by the Organization for the Advancement of Structures Standards (OASIS). The specification is called Web Services Security: SOAP Message Security 1.0 (WS-Security 2004).

- Organization for the Advancement of Structured Information Standards (OASIS)
- Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)

XML Encryption Syntax and Processing

XML Encryption Syntax and Processing specifies a process for encrypting data and representing the result in XML. The data can be arbitrary data (including an XML

document), an XML element, or XML element content. The result of encrypting data is an XML Encryption element that contains or references the cipher data.

XML Encryption Syntax and Processing is a recommendation of the World Wide Web Consortium (W3C):

- World Wide Web Consortium (W3C)

The XML Encryption Syntax and Processing recommendation is published at:

- XML Encryption Syntax and Processing

XML-Signature Syntax and Processing

XML-Signature Syntax and Processing specifies the processing rules and syntax for XML digital signatures.

XML digital signatures provide integrity, message authentication, and signer authentication services for data of any type, whether located in the XML that includes the signature or elsewhere.

The recommendation for XML-Signature Syntax and Processing is published by the World Wide Web Consortium (W3C):

- World Wide Web Consortium (W3C)

The XML-Signature Syntax and Processing recommendation is published at:

- XML-Signature Syntax and Processing

WebSphere Message Broker compliance with Web services standards

WebSphere Message Broker complies with the supported Web services standards and specifications, in that you can generate and deploy Web services that are compliant.

However, WebSphere Message Broker does not enforce this compliancy. For example, for support of the WS-I Basic Profile 1.1 specification, you can apply additional qualities of service to your Web service that might break the interoperability outlined in this Profile.

The topics in this section describe how WebSphere Message Broker complies with Web services standards.

- "How WebSphere Message Broker complies with Web Service Security specifications"

How WebSphere Message Broker complies with Web Service Security specifications

WebSphere Message Broker conditionally complies with Web Services Security: SOAP Message Security and related specifications by supporting the following aspects.

Compliance with Web Services Security: SOAP Message Security

Security header

The `<wsse:Security>` header provides a mechanism, in the form of a SOAP actor or role, for attaching security-related information that is targeted at a specific recipient. The recipient can be the ultimate recipient of the message or an intermediary. The following attributes are supported in WebSphere Message Broker:

- S11:actor (for an intermediary)
- S11:mustUnderstand
- S12:role (for an intermediary)
- S12:mustUnderstand

Security tokens

The following security tokens are supported in the security header:

- Username and password
- Binary security token (X.509 certificate)

Token references

A security token conveys a set of claims. Sometimes these claims are elsewhere and need to be accessed by the receiving application. The `<wsse:SecurityTokenReference>` element provides an extensible mechanism for referencing security tokens. The following mechanisms are supported:

- Direct reference
- Key identifier
- Key name
- Embedded reference

Signature algorithms

This specification builds on XML Signature and therefore has the same algorithm requirements as those specified in the XML Signature specification. WebSphere Message Broker supports the signature algorithms as shown in the following table.

Algorithm type	Algorithm	URI
Digest	SHA1	`http://www.w3.org/2000/09/xmldsig#sha1`
Signature	DSA with SHA1 (validation only)	`http://www.w3.org/2000/09/xmldsig#dsa-sha1`
Signature	RSA with SHA1	`http://www.w3.org/2000/09/xmldsig#rsa-sha1`
Canonicalization	Exclusive XML canonicalization (without comments)	`http://www.w3.org/2001/10/xml-exc-c14n#`

Signature signed parts

WebSphere Message Broker allows the following SOAP elements to be signed:

- The SOAP message body
- The identity token (a type of security token) that is used as an asserted identity

Encryption algorithms

The data encryption algorithms that are supported are shown in the following table.

Algorithm	URI
Triple Data Encryption Standard algorithm (Triple DES)	`http://www.w3.org/2001/04/xmlenc#tripledes-cbc`
Advanced Encryption Standard (AES) algorithm with a key length of 128 bits	`http://www.w3.org/2001/04/xmlenc#aes128-cbc`

Algorithm	URI
Advanced Encryption Standard (AES) algorithm with a key length of 192 bits	`http://www.w3.org/2001/04/xmlenc#aes192-cbc`
Advanced Encryption Standard (AES) algorithm with a key length of 256 bits	`http://www.w3.org/2001/04/xmlenc#aes256-cbc`

The key encryption algorithm that is supported is shown in the following table.

Algorithm	URI
Key transport (public key cryptography) RSA Version 1.5	`http://www.w3.org/2001/04/xmlenc#rsa-1_5`

Encryption message parts
WebSphere Message Broker allows the following SOAP elements to be encrypted:

- The SOAP body

Timestamp
The `<wsu:Timestamp>` element provides a mechanism for expressing the creation and expiration times of the security semantics in a message. WebSphere Message Broker tolerates the use of timestamps within the Web services security header on inbound SOAP messages.

Error handling
WebSphere Message Broker generates SOAP fault messages using the standard list of response codes listed in the specification.

Compliance with Web Services Security: Username Token Profile 1.1

The following aspects of this specification are supported:

Password types
Text

Token references
Direct reference

Compliance with Web Services Security: X.509 Certificate Token Profile 1.1

The following aspects of this specification are supported:

Token types

- X.509 Version 3: Single certificate.
- X.509 Version 3: X509PKIPathv1 without certificate revocation lists (CRL).
- X.509 Version 3: PKCS7 with or without CRLs. The IBM Software Development Kit (SDK) supports both. The Sun Java Development Kit (JDK) supports PKCS7 without CRL only.

For more information, refer to Web Services Security X.509 Certificate Token Profile.

Token references

- Key identifier - subject key identifier

- Direct reference
- Custom reference - issuer name and serial number

Aspects that are not supported

The following items are not supported in WebSphere Message Broker:

- Validation of Timestamps for freshness.
- Nonces.
- Web services security for SOAP attachments.
- Security Assertion Markup Language (SAML) token profile, WS-Security Kerberos token profile, and XrML token profile.
- Web Services Interoperability (WS-I) Basic Security Profile.
- XML enveloping digital signature.
- XML enveloping digital encryption.
- The following transport algorithms for digital signatures:
  - XSLT: `http://www.w3.org/TR/1999/REC-xslt-19991116`.
  - SOAP Message Normalization. For more information, refer to http://www.w3.org/TR/2003/NOTE-soap12-n11n-20031008.
- The Diffie-Hellman key agreement algorithm for encryption. For more information, refer to Diffie-Hellman Key Values.
- The following canonicalization algorithm for encryption, which is optional in the XML encryption specification:
  - Canonical XML with or without comments
  - Exclusive XML canonicalization with or without comments
- The digest password type in the Username Token Version 1.0 Profile specification.

Message flows for Web services

Message flows that need to work with Web services can use either the SOAP domain or one of the XML domains.

The following topics describe both types of flow.

- "SOAP domain message flows"
- "XML domain message flows" on page 822

The following topic describes fundamental scenarios. Any use of WS-Addressing or WS-Security requires use of the SOAP domain, but otherwise these scenarios apply to both types of message flow.

- "Web services scenarios" on page 827

SOAP domain message flows

SOAP domain message flows use SOAP nodes, WSDL, and a common logical tree format which is independent of the exact format of the Web service message.

The following nodes are provided for use in the SOAP domain:

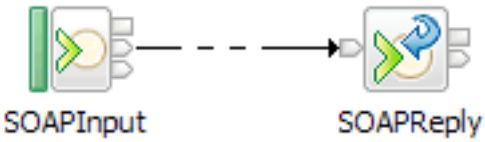
- "SOAPInput node" on page 1194
- "SOAPReply node" on page 1202
- "SOAPRequest node" on page 1204
- "SOAPAsyncRequest node" on page 1172
- "SOAPAsyncResponse node" on page 1181

The following nodes can also be used to simplify SOAP payload processing in a message flow; these nodes are not specific to the SOAP domain.

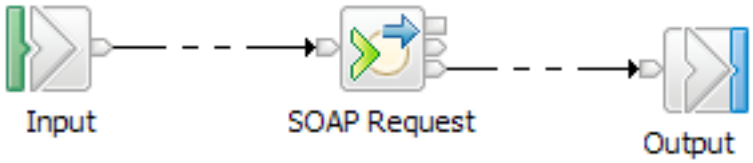
- "SOAPEnvelope node" on page 1186
- "SOAPExtract node" on page 1189

The SOAP nodes are used together in the following basic patterns:

- As a Web service provider, for example:

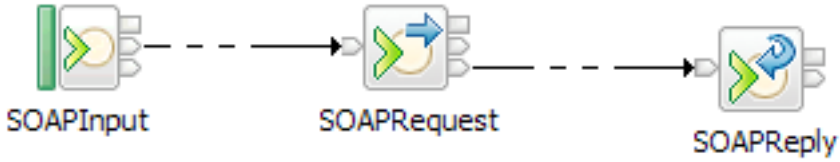


- As a Web service consumer, for example:



Or:

- As a Web service facade, for example, by combining the provider and consumer scenarios:



You can use the SOAPExtract and SOAPEnvelope nodes in conjunction with these patterns to respectively extract the SOAP payload and rebuild a SOAP envelope.

The main SOAP domain nodes are configured by WSDL, and a prerequisite for a SOAP domain message flow is a message set containing deployable WSDL. To create a message set containing deployable WSDL, either import existing WSDL or generate WSDL from an existing message set. For more information about creating a new message definition from WSDL, see Importing from WSDL. For more information about generating WSDL from an existing message set, see WSDL generation.

Alternatively you can create a new message set and skeleton message flow in one step using the procedure described in "Creating an application based on WSDL or XSD files" on page 170.

The WSDL then appears in the workbench under **Deployable WSDL** below the message set, although if you have selected Hide Categories on the message set, the category heading itself is not shown.

Deployable WSDL can then be used to configure SOAP nodes. You can do this by dragging the WSDL resource onto the node or by selecting the required WSDL resource from the node properties.

Alternatively a new skeleton message flow can be created by dragging and dropping the WSDL on to a blank message flow editor canvas.

The WSDL is deployed with your completed message flow, enabling the broker to raise exceptions if a Web service message does not correspond to the specified WSDL description.

The SOAP domain uses a common logical tree format which is independent of the exact format of the Web service message. For details of the SOAP tree format, see "SOAP tree overview" on page 98. Useful WSDL information is included in the logical tree under `SOAP.Context`.

Example usage of WS-Addressing

Set up a sample message flow by using WS-Addressing, and test the flow.

Complete the steps in the following set of topics.

1. Build the main message flow that includes SOAP nodes, a Filter node, and a Mapping node by following the instructions in "Building the main message flow."
2. Build a logging message flow, so that you can send a reply to an address different from the originating client, by following the instructions in "Building the logger message flow" on page 819.
3. Deploy the message flows by following the instructions in "Deploying the message flows" on page 820.
4. Test the message flows, by using a tool that uses HTTP. This task illustrates that the contents of the SOAP message determine where the replies are routed. Follow the instructions in "Testing the message flows" on page 821.

Building the main message flow:

You can construct a sample main message flow to use with WS-Addressing.

These steps are the first in a set of instructions on setting up your system to use WS-Addressing with WebSphere Message Broker; they explain how to set up a message flow to use this feature. This topic describes how to construct a sample main message flow when using WS-Addressing.

1. Switch to the Broker Application Development perspective.
2. Create message flow and message set projects using the Start from WSDL and/or XSD files wizard. See "Creating an application based on WSDL or XSD files" on page 170 for instructions.
3. Select the WSDL file that you need under **Deployable WSDL** from the Active Working Set:
   a. Drag the mouse pointer to the message flow editor.
   b. Release the left mouse button. The Configure New Web Service Usage wizard starts.
   c. See "Creating an application by using the Configure New Web Service Usage wizard" on page 173 for further information. Ensure that you select:
      - **Expose message flow as Web service** on page one of the wizard.
      - **SOAP nodes** on page two of the wizard.

When you select **Finish** a skeleton message flow is generated, consisting of the following nodes:
- SOAPInput node that is pre-configured for the Web service
- SOAPExtract node to remove the SOAP envelope from the incoming message
- SOAPReply node

4. Select the **Construction** folder on the message flow palette to display the contents.
5. Select a Trace node and move the mouse to the right of the SOAPExtract node.
   a. Click the left mouse button to add the node to the message flow. The name is selected automatically.
   b. Press **Enter** to accept the default name.
   c. Wire the submitPORequest terminal of the SOAPExtract node to the In terminal of the Trace node.
6. Select the Trace node to display the properties.
   a. Use the menu to set **Destination** to `File`
   b. Set the **File path** that you require.
   c. Enter the **Pattern** that you require.
7. Expand the **Routing** folder on the palette and select **Filter**.
8. Add the Filter node to the right of the Trace node.
   a. Type the name for the node that you require and press **Enter**.
   b. Wire the Out terminal of the Trace node to the In terminal of the Filter node.
9. Select the Filter node to display the properties.
   a. Enter the **Data source** name that you require.
   b. Change the name of **Filter expression** to the name that you selected for the Filter node.
   c. Clear the **Throw exception on database error** check box.
10. Double-click the Filter node to open the ESQL editor. Create or change the ESQL for the node; for more information, see "Creating ESQL for a node" on page 315 and "Modifying ESQL for a node" on page 318.
11. Expand the **Transformation** folder on the palette and select a Mapping node.
12. Add the Mapping node to the right of the Filter node.
    a. Type the name for the node that you require and press **Enter**.
    b. Wire the True terminal of the Filter node to the In terminal of the Mapping node.
    c. Wire the Out terminal of the Mapping node to the In terminal of the Reply node.
13. Select the Mapping node to display the properties, and change the name of **Mapping routine** to the name that you selected for the Mapping node.
14. Double-click the Mapping node to open the mapping editor.
    a. Select **submitPORequest** as the map source.
    b. Select **SOAP_Domain_msg** as the map target.
    c. Click **OK**
    d. Click **OK** on the tip that is displayed to open the mapping editor.

    See "Creating a message map file from a Mapping node" on page 552 for further information.

15. Select **Properties** in both the **source** and **target** pane, right-click, and click **Map by Name**.
    a. Map the source properties to the target properties using drag-and-drop mapping. The Map by Name dialog box appears.
    b. Click **OK** to perform the mapping.
16. Expand **SOAP_Domain_Msg**, then **Body** and **message items** in the target pane.
17. Right-click **Wildcard Message** in the target pane, and click **Create new Submap**.
    a. Expand **Wildcard**.
    b. Scroll down and click **submitPOResponse**.
    c. Click **OK** to create the submap.
18. Use drag-and-drop mapping to select the items that you need in the **Source** pane.
19. Select the first item that you need in the **Target** pane, right-click, and **Enter Expression**. Enter the value that you require in the expression editor and press **Enter** to complete the mapping.

    Repeat the above steps for all the items that you require in the **Target** pane, and save the submap and map by pressing **Ctrl+S**.
20. Expand the **Construction** folder on the message flow editor and select a Throw node.
21. Add the Throw node above the Mapping node
    a. Type the name for the node that you require and press **Enter**.
    b. Wire the False and Unknown terminals of the Filter node to the In terminal of the Throw node.
22. Select the Throw node and, in the Node Properties pane, enter the text that you require in **Message text**.
23. Select the SOAPInput node to display the Node Properties.
    a. Select the **WS Extensions** tab.
    b. Select **Use WS-Addressing** .
24. Save the message flow.

Building the logger message flow:

This is the second of a set of instructions on setting up your system to use WS-Addressing with WebSphere Message Broker, and illustrates the use of a reply being sent to an address other than the originating client.

This topic describes the construction of a sample logger message flow when using WS-Addressing. This flow echoes back the input while creating a trace entry in a file to indicate that the flow has been invoked.

1. Switch to the Broker Application Development perspective.
2. Select the message flow name that you used in “Building the main message flow” on page 817
3. Press the right mouse button and select **New->MessageFlow**.
    a. Enter the name that you require for this message flow; for example, `Logger`.
    b. Press **Finish** to create the flow.
4. Select the **HTTP** folder on the message flow palette to display the contents.
5. Select an **HTTPInput** node and move the mouse to the left side of the canvas.

a. Click the left mouse button to add the node to the message flow and enter the name `Logger`.
b. Press **Enter** to finish.

6. Select the **HTTPReply** node from the palette and move the mouse to the right of the HTTPInput node, leaving room for a node in between.
   a. Click the left mouse button to add the node to the message flow and enter the name that you require; for example, `Logger`.
   b. Press **Enter** to finish.
7. Select the **Construction** folder on the message flow palette to display the contents.
8. Select a **Trace** node and move the mouse to the right of the HTTPInput node.
   a. Click the left mouse button to add the node to the message flow and enter the name that you require; for example `Trace`.
   b. Press **Enter**.
   c. Wire the `out` terminal of the HTTPInput node to the `In` terminal of the Trace node.
   d. Wire the `out` terminal of the Trace node to the `In` terminal of the HTTPReply node.
9. Select the HTTPInput node to display the properties. In the **Basic** tab:
   a. Enter the **Data source** name that you require.
   b. Change the name of the input node `Logger` as the **Path suffix for URL**.
10. Select the **Input Message Parsing** tab and select `XMLNSC` as the **Message domain**.
11. Select the Trace node to display the properties.
    a. Set **Destination** to `File`
    b. Set the **File path** that you require.
    c. Enter the **Pattern** that you require.
12. Save the message flow.

Deploying the message flows:

This is the third of a set of instructions on setting up your system to use WS-Addressing with WebSphere Message Broker, and illustrates the deployment of the message flows.

This topic describes the deployment of the message flows you have already constructed.

1. Switch to the Broker Administration perspective.
2. Select the **LocalProject** project under **Broker Archives** in the navigator pane.
   a. Press the right mouse button.
   b. Select **New->message Broker Archive**.
   c. Use the drop down menu to change the **Project** to `LocalProject`.
   d. Enter the name you selected for the main message flow described in "Building the main message flow" on page 817 in **Name**.
   e. Press **Finish** to open the Broker Archive Editor.
3. In the Broker Archive Editor:
   a. In the **Filter working set** list box select the working set name that you used for the main message flow.

b. In **Message Flows** select the message flow names that you used for the main message flow and logger message flow.
c. In **Message Sets** select the message set name that you used for the main message set.
d. Press **Build broker archive** and confirm that the build operation was successful.
e. Save the updated broker archive.

4. To deploy the message flows to the default execution groups:
   a. Select the name of the broker archive file with the name of the message flow that you used for the main message flow.
   b. Press the right mouse button.
   c. Select **Deploy File** from the menu.
5. Select the **default** execution group and press **OK** to start the deployment. Ensure that you receive a successful response message, and press **OK** to dismiss the information dialog.
6. Use the Event log to confirm that the deploy operation was successful:
   a. Double click the **Event Log** entry on the **Domains** tab to open the Event log editor.
   b. Confirm that the deployment was successful.

Testing the message flows:

This is the fourth of a set of instructions about how to set up your system to use WS-Addressing with WebSphere Message Broker; these instructions illustrate how to test the message flows.

This topic describes how to test the message flows that you have already constructed. In this scenario, you use a tool that uses HTTP protocol rather than WebSphere MQ protocol. You can use any tool that has the facilities that are described in the following procedure.

1. Start the tool and select `http://localhost:nnnn/`, where *nnnn* is the address of the port that you are using.
2. Set the URL to the host, port, and selection for the deployed flow. The SOAPInput nodes listen on a different port from the HTTP nodes, and the listener is built into the execution group rather than using a different process.
3. The SOAPInput node expects a SOAPAction entry in the HTTP headers, therefore you must add one.
   a. Click **Add New Header**.
   b. Enter the **Value** part of the header. The value must match the `SOAPAction` attribute of the `SOAP:operation` element in your code.
   c. Select **New Header** in the **Name** pane.
   d. Change the name from **New Header** to `SOAPAction` and click **Enter**.
4. Select **Load File** and go to the directory that contains the XML file you want to use.
5. Select the file and click **Open**. Note the following conditions:
   - If the message does not include any WS-Addressing entries, the `ReplyTo` and `FaultTo` locations default to `anonymous`. This means that the results are returned on the original client connection.
   - If the message includes a WS-Addressing header (`ReplyTo`) with a value of `anonymous`, the reply is returned to the original client by using the original TCP/IP connection.

- If the message includes a WS-Addressing header with a value of `FaultTo` explicitly included, the reply is returned to that address rather than the default of using the location that was specified in the `ReplyTo` header.

6. Click **Send** to test the flow. The result appears in the right pane.

XML domain message flows

If you are not using the SOAP domain, your message flow must take account of the bitstream format of the Web service messages with which you are working. A different logical tree format is used by each domain.

If the messages are SOAP, you can use either the XMLNSC domain or the MRM XML domain. Both domains offer validation. The XMLNSC domain is more efficient, while the MRM XML domain can be useful if you have specific message transformation requirements (for example, if your message flow also uses binary data formats).

If the messages use MIME (for example, SOAP with Attachments or MTOM), you can use the MIME domain. In this case, your message flow typically needs to identify at least the MIME part that corresponds to the SOAP payload, then explicitly parse this part by using the XMLNSC or MRM domain.

In the SOAP domain, WSDL is used to configure your nodes automatically with the appropriate endpoint information. If you are not using the SOAP domain, select and configure the transport nodes manually. Typical WSDL bindings are:

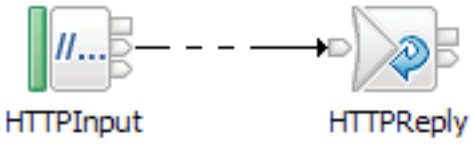
- SOAP/HTTP; in which case, implement a flow by using HTTP nodes. Use the HTTPInput and HTTPReply nodes if a flow implements a Web service, or use the HTTPRequest node if a flow calls a Web service.
- SOAP/JMS; where you implement a flow by using JMS or MQ nodes.

You can configure message flows that receive input messages from clients by using one transport, and interact with a Web service or legacy application by using another.

You can propagate a message to more than one location. For example, the Web service response to be returned to a client by an HTTPReply node might first be sent to an auditing application by using an MQOutput node, after making any required adjustments to the message headers.

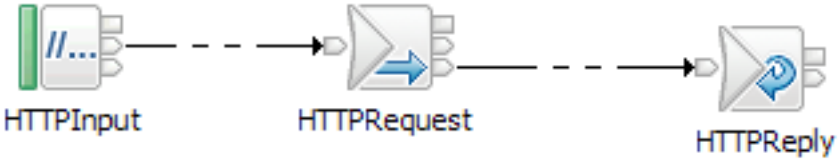
Nodes are used together in the following basic patterns, by using HTTP nodes as example transports:

- As a Web service provider, for example:



- As a Web service consumer, for example:

- As a Web service facade, for example:



If required, you can use the SOAPExtract and SOAPEnvelope nodes in conjunction with these patterns to respectively extract the SOAP payload and rebuild a SOAP Envelope.

To enable your message flow to validate messages, deploy an appropriate message set with the flow. An appropriate message set is created either by importing existing WSDL or by generating WSDL from an existing message set. For details about importing existing WSDL, see Importing from WSDL. For details about generating WSDL from an existing message set, see WSDL generation.

You can also create a new message set and flow based on existing WSDL or XSD files; for details, see “Creating an application based on WSDL or XSD files” on page 170.

The generated message set contains message definitions for the relevant SOAP Envelope version and for the XML payload data defined by the WSDL. In the XMLNSC or MRM XML domains, messages can be validated against the message set; for details, see “Validating messages” on page 204.

Working with HTTP flows

Read this information if you are using HTTP message flows to interact with Web services.

You might find it useful to read this information with the “Web services scenarios” on page 827 section.

- Using secure connections with HTTPS
- Setting the HTTP Status Code for a reply
- Using LocalEnvironment.Destination.HTTP.RequestIdentifier
- Setting the HTTPRequest node URL dynamically
- Setting Generate default HTTP headers from reply or response for the HTTPReply node
- Setting Generate default HTTP headers from input for the HTTPRequest node
- Collecting HTTPListener trace if you have problems with HTTP

You can use `ProxyConnectHeaders` only with HTTPS (SSL) connections; these headers do not work with HTTP connections.

Using secure connections with HTTPS

Fro information about setting up HTTPS connections, see Implementing SSL authentication.

Setting the HTTP Status Code for a reply

The default HTTP Status Code is 200, which indicates success. If you want a different status code to be returned, take one of the following actions:

- Set your status code in the field Destination.HTTP.ReplyStatusCode in the local environment tree (correlation name OutputLocalEnvironment). This field overrides any status code that is set in an HTTPResponseHeader header. This action is the preferred option, because it provides the greatest flexibility.
- Set your status code in the field X-Original-HTTP-Status-Code in the HTTPReplyHeader header.
- Set your status code in the field X-Original-HTTP-Status-Code in the HTTPResponseHeader header. This option is useful if you include an HTTPRequest node before the HTTPReply node in your flow, because the HTTPResponseHeader header is created for you. In this scenario, an HTTPResponseHeader header has been created in the logical tree, representing the HTTP headers in the response from another Web service. If you have selected the Generate default HTTP headers from reply or response property on the HTTPReply node, values for the response header are set as default values when the reply message is created.

Using LocalEnvironment.Destination.HTTP.RequestIdentifier

When the HTTPInput node receives an input request message, it sets the local environment field Destination.HTTP.RequestIdentifier to a unique value that identifies the Web service client that sent the request. You can refer to this value, and you can save it to another location if appropriate.

For example, if you design a pair of message flows that interact with an existing WebSphere MQ application (as described in "Broker calls existing Web service" on page 827), you can save the identifier value in the request flow, and restore it in the reply flow, to ensure that the correct client receives the reply. If you use this technique, you must not change the data, and you must retain the data as a BLOB.

The HTTPReply node extracts the identifier value from the local environment tree and sets up the reply so that it is sent to the specific client. However, if you are using an HTTPReply node in a flow that does not have an HTTPInput node, and this field has been deleted or set incorrectly, message BIP3143S is issued.

If you design a message flow that includes both an HTTPInput and an HTTPReply node, the identifier value is set into the local environment by the HTTPInput node, but the HTTPReply node does not use it. Therefore, if your message flow includes both HTTP nodes and a Compute node in the same flow, you do not have to include the local environment tree when you specify which components of the message tree are copied from input message to output message by the Compute node (the Compute mode property).

Setting the HTTPRequest node URL dynamically

You can set the property Default Web service URL on the HTTPRequest node to determine the destination URL for a Web service request. You can configure a Compute node before the HTTPRequest node within the message flow to override the value set in the property. Code ESQL that

stores a URL string in LocalEnvironment.Destination.HTTP.RequestURL; the HTTPRequest node retrieves and uses the URL string in place of the node property value.

Although you can also set the request URL in the special header X-Original-HTTP-URL in the HTTPRequestHeader header section of the request message (which overrides all other settings) in a Compute node, use the local environment tree content for this purpose for greater flexibility.

Setting Generate default HTTP headers from reply or response for the HTTPReply node

If you select Generate default HTTP headers from reply or response in the HTTPReply node properties, the node includes a minimum set of headers in the response that is sent to the Web service client.

To set headers explicitly, create them in an HTTPReplyHeader header. For example, a Compute node propagates a message in the XMLNS domain and modifies the Content-Type as follows:

```
CALL CopyMessageHeaders();
SET OutputRoot.HTTPReplyHeader."Content-Type" = 'text/xml';
SET OutputRoot.XMLNS = InputRoot.XMLNS;
```

Do not use the ContentType property to set the Content-Type unless you are working in the MIME domain. The ContentType property is intended to set the value of Content-Type used in MIME.

The full set of HTTP headers used in the reply is built by selecting the headers using the algorithm defined in the following steps:

1. Select one or more headers in an HTTPReplyHeader header.
2. If no Content-Type header is yet defined, create one by using a non-empty value in the ContentType property.
3. Select one or more headers in an HTTPResponseHeader header (an HTTPResponseHeader header is propagated on return from an HTTPRequest node).
4. If no Content-Type header is yet defined, create one with the default value `text/xml; charset=`*`ccsid of the message body`*.
5. Create or overwrite the Content-Length header.

Attention: The HTTPReply node always rewrites the Content-Length header, even if you have cleared Generate default HTTP headers from reply or response. This action ensures that the content is correct.

If an HTTPReplyHeader header section existed in the message received by the HTTPReply node, and the Output terminal of the HTTPReply node is connected, the HTTPReplyHeader header section is updated with all changed or added values.

Setting Generate default HTTP headers from input for the HTTPRequest node

If you select Generate default HTTP headers from input in the HTTPRequest node properties, the node includes a minimum set of headers in the request that is sent to the server.

To explicitly set headers, create them in an HTTPRequestHeader header. For example, a Compute node propagating a message in the XMLNS domain can modify the Content-Type as follows:

```
CALL CopyMessageHeaders();
SET OutputRoot.HTTPRequestHeader."Content-Type" = 'text/xml';
SET OutputRoot.XMLNS = InputRoot.XMLNS;
```

Do not use the ContentType property to set the Content-Type unless you are working in the MIME domain. The ContentType property is intended to set the value of Content-Type used in MIME.

The full set of HTTP headers used in the request is built by selecting the headers using the algorithm defined in the following steps:

1. Set the Host header, based on either the request URL or the incoming HTTPRequestHeader header section of the message.
2. Select one or more headers in an HTTPRequestHeader header.
3. If no Content-Type header is yet defined, create one by using a non-empty value in the ContentType property.
4. Select one or more headers in an HTTPInputHeader header (an HTTPInputHeader header is created automatically by an HTTPInput node).
5. If no Content-Type header is yet defined, create one with the default value `text/xml; charset=`*`ccsid of the message body`*
6. If no SOAPAction header is yet defined, create one with the default value `''`.
7. Create or overwrite the Content-Length header.

Attention: The HTTPRequest node always rewrites the Content-Length header, even if you have cleared Generate default HTTP headers from input or request. This action ensures that the content is correct.

If an HTTPRequestHeader header exists in the received message, the HTTPRequestHeader header is updated with all changed or added values.

Collecting HTTPListener trace if you have problems with HTTP

If you have problems with processing HTTP messages:

1. Start trace for the execution group; for example;
   ```
   mqsichangetrace WBRK_BROKER -t -e default -l debug
   ```
2. Start trace the HTTPListener component in one of the following two ways:
   - Trace all broker components:
     a. Run the mqsichangetrace command to start trace with the following options:
        ```
        mqsichangetrace component -t -b -l debug
        ```
        where *component* is the broker name.
     b. Retrieve the HTTPListener trace log by using the mqsireadlog command with the HTTPListener qualifier for the **-b** parameter. For example:
        ```
        mqsireadlog brokerName -t -b httplistener -f -o listenertrace.xml
        ```
     c. Format the trace log by using the mqsiformatlog command to view its contents.
   - Trace only the HTTPListener component:
     a. Run the mqsichangeproperties command to start trace with the following options:

```
mqsichangeproperties brokerName -b httplistener -o HTTPListener
 -n traceLevel -v debug
```

b. Retrieve and format the HTTPListener trace log as shown in the previous example.

Save the trace output so that you can send it to the IBM Support Center, if requested.

Turn trace off when you have finished collecting information to avoid affecting the performance of the broker.

Web services scenarios

These common Web services scenarios are organized according to the role that is played by the broker.

A key consideration is whether a WSDL description for the Web service already exists.

In the first two scenarios, the WSDL description exists and is imported and used by the message flow.

In the remaining two scenarios, the WSDL description is generated in an existing message set. The WSDL is used by the message flow and might also be exported for use by an external client.

These scenarios are generic and can be implemented by using the SOAP domain, or an appropriate non-SOAP domain (XMLNSC, MRM, MIME) and basic transport nodes. If you need to use WS-Addressing or WS-Security for a particular implementation, use the SOAP domain.

You want the broker to invoke an existing Web service:
See "Broker calls existing Web service"

You want the broker to expose an application as a previously defined Web service:
See "Broker implements existing Web service interface" on page 836

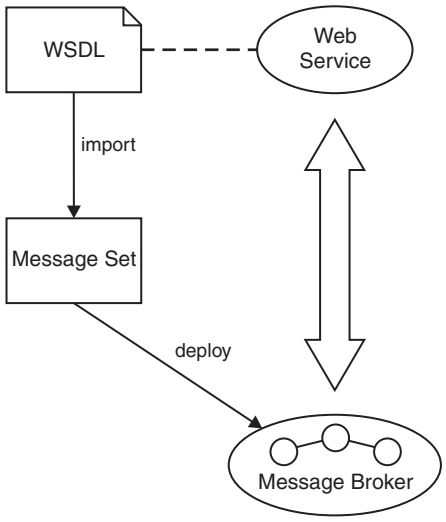
You want the broker to expose an application as a new Web service:
See "Broker implements new Web service interface" on page 832

You want the broker to expose a Web service to a non-Web service client:
See "Broker implements non-Web-service interface to new Web service" on page 840

Broker calls existing Web service

In this scenario, the broker calls an existing Web service implementation. The WSDL for the Web service already exists, and is imported to create a message set.

A message flow based on this message set sends a Web service request and receives the response, for example by using a SOAPRequest node.



Key to symbols:

Executable	File	Message set	Message flow
association	design time action. For example, import or deploy	design time action involving an external toolkit. For example, generating a Web service client.	run time interaction. For example, message exchange

Possible uses

- You want to call a Web service to do some processing as part of your message flow.
- You have an existing Web service and you want to provide a different interface to it. This could be an alternative Web services interface or a WebSphere MQ interface.
- You have an existing Web service and you want to change its implementation in some way without changing its interface; that is, the broker acts as an intermediary to the Web service. For instance a message flow could be used to enable auditing, or to transparently propagate the Web service response to another application.

Design steps

1. Import WSDL to create a message set containing definitions for the SOAP messages described by the WSDL.
2. Create a message flow to invoke the Web service. If the SOAP domain is used, the message flow uses a SOAPRequest node, SOAPAsyncRequest node, or a SOAPAsyncResponse node. The nodes are configured by using the WSDL imported in Step 1. If required, you can create a skeleton flow from scratch by dropping the WSDL onto a blank message flow editor canvas. If you use the SOAP domain, you must create the message flow by using transport nodes,

and an XML or MIME domain. For example, if the WSDL binding specifies HTTP transport, and the request message is SOAP, you can use an HTTPRequest node with the XMLNSC domain. You can then configure the node manually with the endpoint information for the Web service.

3. Build a broker archive file for deployment. The broker archive file contains your message flow and the message set that contains the imported WSDL. The SOAP domain always requires the WSDL to be deployed, because messages are verified against it at run time; also WSDL information is included in the logical tree. The message set includes XML Schema definitions that can be used for message validation in the SOAP, XMLNSC, or MRM domains.

At run time

Your message flow creates an appropriately formatted Web service request, invokes the Web service, and parses the Web service response. If you use the SOAP domain, your message flow uses the SOAP logical tree model. If you do not use the SOAP domain, your message flow uses the logical tree for your selected domain; for example, you use the MIME domain if your Web service messages use SOAP with Attachments.

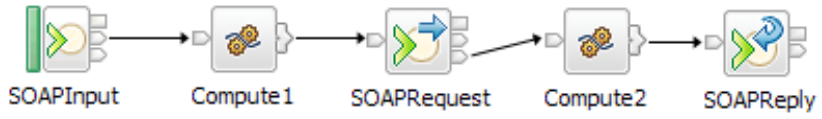
Example 1

Web service intermediary

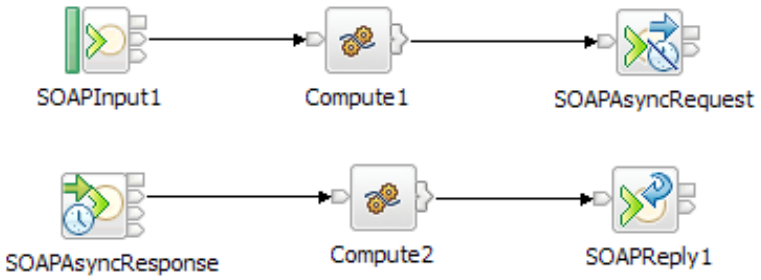
In this example a client application uses a Web service called Account, which is made available by another organization. The client is widely distributed in your company. The client uses an operation called getBalance, but the Account service is being modified to change the definition of the getBalance operation. You can construct message flows to provide an interface to the Account service, instead of modifying the client. The message flows can call the Account service to do the work, and the new Web service delegates to the original Web service. The client can now continue to use the Account service, by using the new message flows.

In the examples of typical message flow patterns shown here, the intermediate request node calls the Account service:

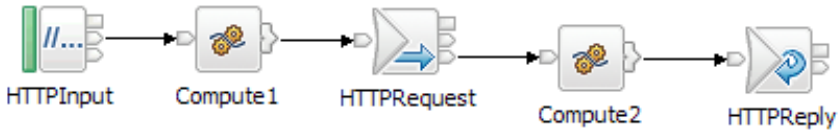
1. Using SOAPInput, SOAPRequest, and SOAPReply nodes:



2. Using SOAPInput, SOAPAsyncRequest, SOAPAsyncResponse, and SOAPReply nodes:



3. Using HTTPInput, HTTPRequest, and HTTPReply nodes:



In the message flows in the example, Compute1 modifies the original getBalance message as required by the modified Account service, while Compute2 restores the response message to the original format. If you have imported or generated WSDL, you have a message model for the getBalance operation. If you have a message model defined for the getBalance operation, you can use Mapping nodes instead of Compute nodes.

HTTP details

If you use HTTP transport nodes, as shown in the example, you can configure the HTTPRequest node to generate HTTP headers from the headers that are received by the HTTPInput node. This configuration enables cookies and other application-specific headers to be passed through the message flow. The HTTPReply node can be used task to extract headers from the Web service response, to return to the originating client. To create this configuration, select **Generate default HTTP headers from** on both the HTTPRequest and HTTPReply nodes. Typically, you do not need the original request message to generate the reply to the client, and can select **Replace input message with Web service response** on the HTTPRequest node. If you do want to preserve data from the input request, you can store this in the LocalEnvironment in Compute1, and retrieve it in Compute2 for inclusion in the reply.

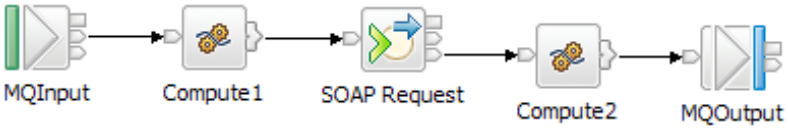
Example 2

Using a Web service

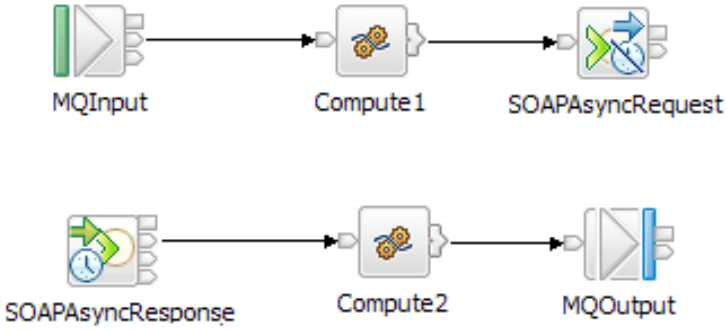
In this example, a WebSphere MQ message flow implements a process for the Human Resource department of your company. As part of this processing, the message flow calls a Web service to retrieve employee ID numbers. Employee ID numbers are maintained in the company's employee directory, which is accessed through a Web service.

In the examples of typical message flow patterns shown here the intermediate request node retrieves the employee ID:

1. Using MQInput, SOAPRequest and MQOutput nodes:



2. Using MQInput, SOAPAsyncRequest, SOAPAsyncResponse, and MQOutput nodes:

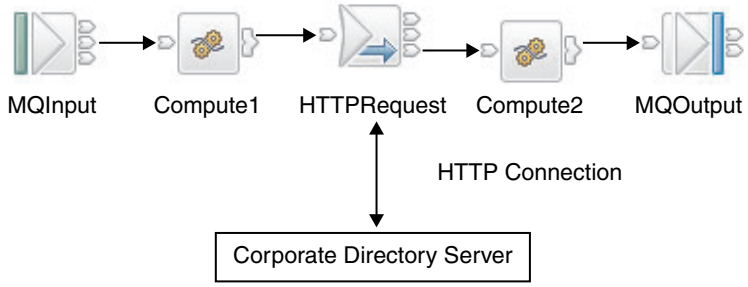


3. Using MQInput, HTTPRequest, and MQOutput nodes:

In the message flows in the example, Compute1 prepares the Web service request message and Compute2 processes the response. For example, by incorporating the employee ID in another message. If you have a message model defined, you can use Mapping nodes instead of Compute nodes in these examples.

HTTP details

If you use HTTP transport nodes, as shown in the example, you typically clear the **Replace input message with Web service response** in the HTTPRequest node properties. The response from the corporate directory server is placed in a temporary location in the message tree. The temporary location is specified in the **Response message location** in tree property in the same node. In Compute2, you can code ESQL to retrieve the result, and update the final message.

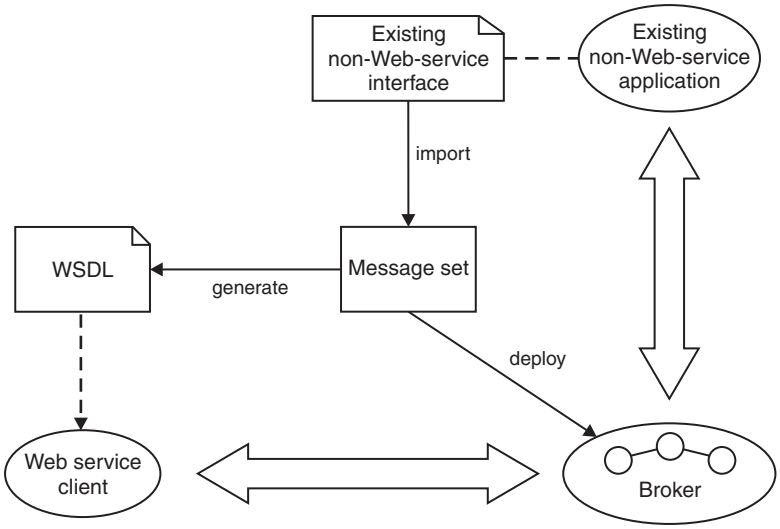


Using the SOAP domain for these scenarios is preferred. For more information about choosing a domain for Web services, see "WebSphere Message Broker and Web services" on page 735.

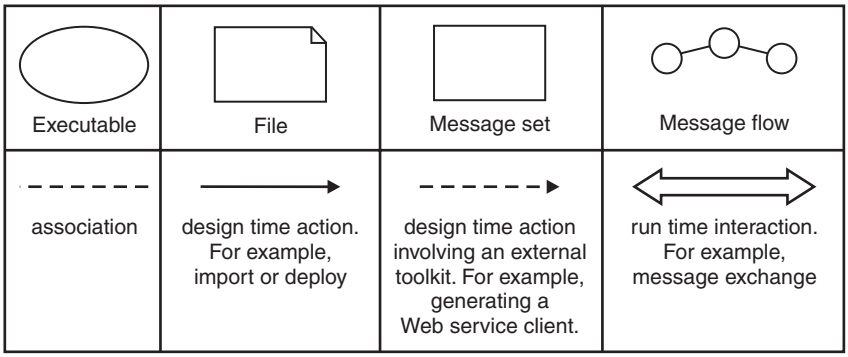
Broker implements new Web service interface

In this scenario, the broker implements a new Web service interface. The WSDL for the Web service is generated from a message set and made available to clients. A message flow based on this WSDL and message set receives a request and then builds a response message by using data obtained from an existing non-Web-service application.

The following diagram shows a message set being created from an interface definition (for example, a header file) of an existing application that is not currently accessible as a Web service. A WSDL file is generated from the message set and exported for use by a Web service client. A message flow that uses the message set and WSDL is created to call the application. The message flow and message set are deployed to a broker, providing a Web service interface to the original application.



Key to symbols:



This scenario is sometimes referred to as a Web service facade. The design of the Web service interface typically involves some regrouping, restriction, or enhancement of the existing interface, and is not constrained by an existing WSDL definition.

Possible uses

- The broker provides a Web services interface to an existing application, optionally providing other mix-in capabilities such as auditing the requests made.
- Over time the implementation can be changed without affecting the interface presented to the Web services client.

Design steps

1. Create a message set for the business messages, possibly by importing an existing interface definition such as a C header file or COBOL copybook.
2. Generate a WSDL definition from the message set.
3. Use a SOAP toolkit such as Rational® Application Developer to create a suitable Web services client based on the WSDL.
4. Develop a message flow to implement the Web service.

At run time

Your message flow receives a Web service request, converts it into a form expected by the existing application and invokes the existing application. The response from the existing application is converted into a valid Web service response.

Example 1

In this example, an existing message flow is modified to provide a Web service. If the existing message flow models its data in a message set, a WSDL definition can be generated from that message set and made available to clients.

Most message flows that currently use WebSphere MQ for input or output can be adapted to support Web services as a replacement or additional protocol.

The following are typical message flow patterns. In each case the input and reply nodes replace or complement the original MQInput and MQOutput nodes. The main part of the flow is understood to do some useful processing.

1. Using SOAPInput and SOAPReply nodes:

2. Using HTTPInput and HTTPReply nodes:

If you use the SOAP domain, the logical tree shape will be different from the original domain and you will need to take account of this in the message flow. If you use the HTTP nodes with the original domain, the logical tree shape does not change. For information about choosing the domain, see "WebSphere Message Broker and Web services" on page 735.

HTTP details

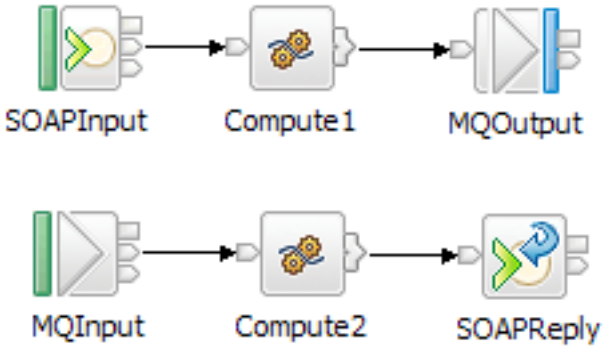
If you use the HTTP nodes, you can configure the HTTPReply node to generate a set of default HTTP headers for the reply message sent to the client. Generating a set of default HTTP headers reduces the modifications that you must make to convert the message flow from one that processes WebSphere MQ messages to a flow that processes HTTP messages.

Example 2

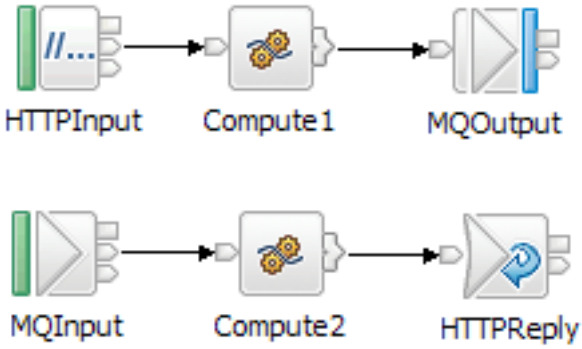
In this example, a message flow is created to provide asynchronous access to a WebSphere MQ application.

The following are typical message flow patterns. In each case the flow receives the Web service request and build the response by using data returned from the application over WebSphere MQ.

1. Using two message flows with SOAPInput and SOAPReply nodes:



2. Using two message flows with HTTPInput and HTTPReply nodes:



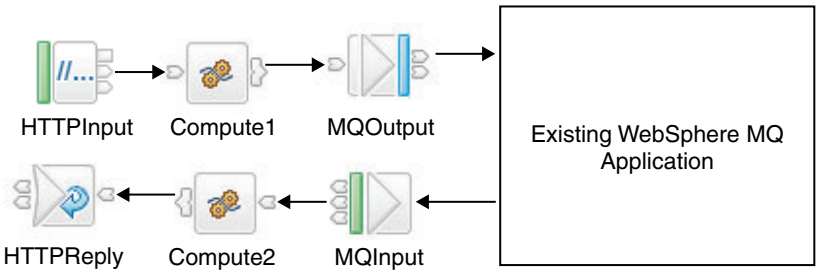
In each case, the first message flow receives inbound requests from a Web service client. The Compute1 node transforms the request and an MQOutput node sends the modified request to the existing application.

In the second message flow, an MQInput node receives the response from the application. The Compute2 node then transforms the message and propagates it to a reply node that responds to the original Web service client.

The Compute1 node must also save some correlation information to be retrieved by the Compute2 node, ensuring that the replies from the WebSphere MQ application are returned to the client that sent the original request.

HTTP details

Using HTTPInput and MQOutput nodes as the outbound message and MQInput and HTTPReply nodes as the response message:



One way to preserve the correlation information is for the Compute1 node to encode the correlation identifier in the outbound message. (Alternatively, the identifier can be stored in a database). The SOAPInput and HTTPInput nodes place the identifier as a field in the local environment tree and the Compute1 node can read and store this value. The location of the identifier differs between the SOAPInput and HTTPInput nodes, as described in the following sections.

SOAP nodes

The Compute2 node reads the SOAP reply identifier from the message and sets LocalEnvironment.Destination.SOAP.Reply.ReplyIdentifier by using this value. The SOAPReply node uses the reply identifier to ensure that the message reaches the correct HTTP client. In the ESQL module for the Compute1 node, include a code statement like the following statement:

```
SET OutputRoot.XMLNS.A.MessageID =
CAST(InputLocalEnvironment.Destination.SOAP.Reply.ReplyIdentifier AS CHARACTER);
```

In the ESQL module for the Compute2 node, include a code statement like the following statement:

```
SET OutputLocalEnvironment.Destination.SOAP.Reply.ReplyIdentifier =
CAST(InputRoot.XMLNS.A.MessageID AS BLOB);
```

HTTP nodes

The Compute2 node reads the HTTP request identifier from the message and sets LocalEnvironment.Destination.HTTP.RequestIdentifier by using this value. The HTTPReply node uses the request identifier to ensure that the message reaches the correct HTTP client. In the ESQL module for the Compute1 node, include a code statement like the following statement:

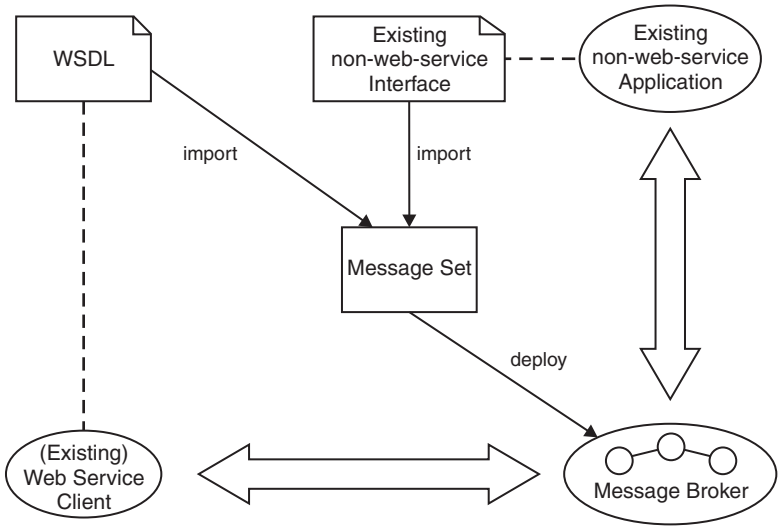
```
SET OutputRoot.XMLNS.A.MessageID =
CAST(InputLocalEnvironment.Destination.HTTP.RequestIdentifier AS CHARACTER);
```

In the ESQL module for the Compute2 node, include a code statement like the following statement:

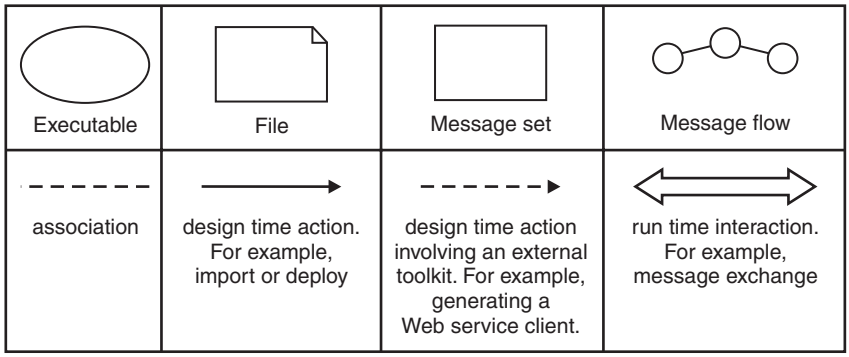
```
SET OutputLocalEnvironment.Destination.HTTP.RequestIdentifier =
CAST(InputRoot.XMLNS.A.MessageID AS BLOB);
```

Broker implements existing Web service interface

In this scenario, the broker implements an existing Web service interface. The WSDL for the Web service already exists, and is imported to create a message set. A message flow based on this message set receives a request, then builds a response message by using data obtained from an existing non-Web-service application.



Key to symbols:



Possible uses

- The broker provides a Web service implementation with a different quality of service from existing implementations.
- The broker provides a migration strategy for the existing implementation.

Design steps

1. Import WSDL to create a message set containing definitions for the SOAP messages described by the WSDL.
2. Adapt the message set for the required existing interface, possibly by importing an existing interface definition such as a C header file or COBOL copybook.
3. Develop a message flow to implement the Web service.

At run time

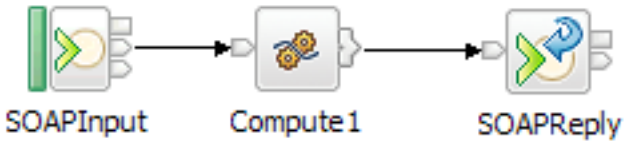
Your message flow receives a Web service request, converts it into a form expected by the existing application and invokes the existing application. The response from the existing application is converted into a valid Web service response.

Example 1

In this example, an existing HTTP Web service client provides information on a particular subject (stock prices or exchange rates, for example). You want to replace this service with an inhouse database lookup solution, but want to make no changes to the clients because these are widely deployed.

Typical message flow patterns are shown in the following examples. In each case the intermediate request node retrieves the information from the database:

1. Using SOAPInput and SOAPReply nodes:



2. Using HTTPInput and HTTPReply nodes:

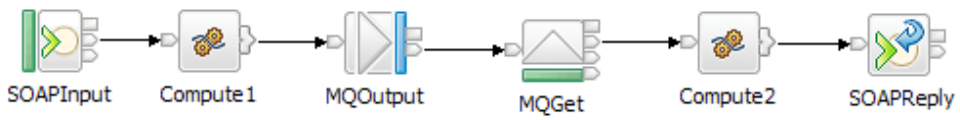
In the flows above, the input node receives the Web service request. Compute1 then retrieves the required information from the database and generates the required Web service response by using this data. The response is returned to the client by the reply node. In the examples you can use Mapping nodes instead of Compute nodes.

Example 2

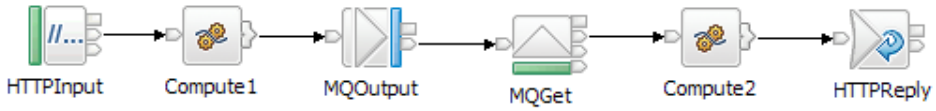
In this example, an existing application is exposed as a Web service, but there is a constraint on the interface with the Web service, because a widely distributed client already uses a similar service that is defined by an existing WSDL definition. The broker offers the same interface to the client, this might be because the original service offers a different quality of service or is to be discontinued.

Typical message flow patterns are shown in the following examples. In each case the message flows receive the Web service request and build the response by using data returned from the application over WebSphere MQ.

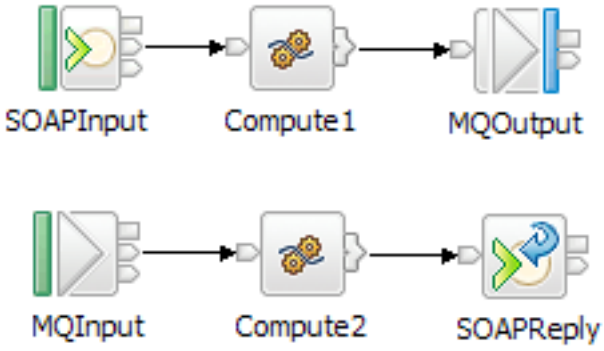
1. Using SOAPInput, SOAPReply and MQGet nodes:



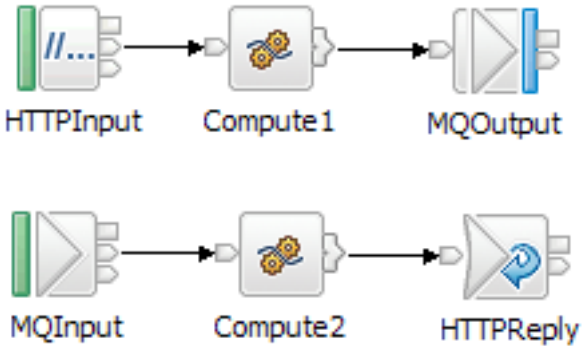
2. Using HTTPInput, HTTPReply and MQGet nodes:



3. Using two message flows with SOAPInput, SOAPReply nodes:



4. Using two message flows with HTTPInput and HTTPReply nodes:



The steps to develop the message flow are:

1. Create a message model for the existing application interface, for example, by importing a C header file for the application.
2. Import an existing WSDL definition for the client.
3. Create a flow by using the message set to implement the Web service interface and mediate with the existing application.

Message flows 1 and 2 show a synchronous call to the application by using MQOutput and MQGet nodes. You can set a timeout in the MQGet node, to allow for failure of the remote application. The request-reply translation is handled in a single transaction enabling simple rollback and recovery. However, each incoming request has to be fully processed and responded to before moving onto the next request. This behavior might affect performance if the application cannot respond

quickly. The message flows shown in examples 3 and 4, show an asynchronous equivalent. In each case the first flow stops after sending the message to the application, and becomes available to handle further requests. However, this scenario requires a correlation context to be saved in the request flow, and restored in the reply flow. You can store the correlation context on a queue, then use an MQGet node in the reply flow to retrieve it. This flow design enables the requests to be dispatched to the application promptly, and replies to be returned in the order that they are received. In the examples you can use Mapping nodes instead of Compute nodes.

The use of the SOAP domain for these scenarios is preferred. For more information about choosing a domain for Web services, see "WebSphere Message Broker and Web services" on page 735.

For more information about the asynchronous request-reply scenario, see "A request-response scenario that uses an MQGet node" on page 230.

The asynchronous request-reply scenario is also detailed in the following sample which can be adapted for Web service usage:

- Coordinated Request Reply

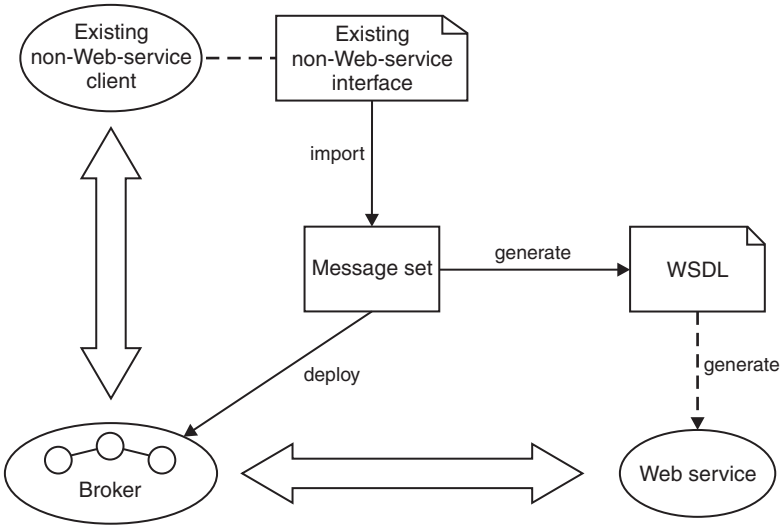
Another Web services scenario is described in the sample:

- HTTP Nodes

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

Broker implements non-Web-service interface to new Web service

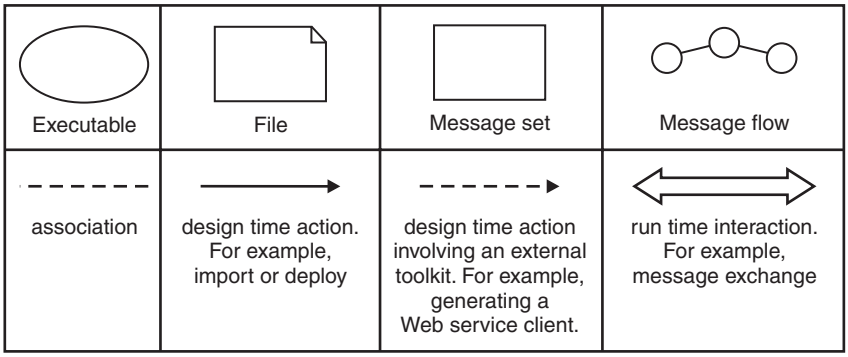
In this Web service scenario, the broker provides compatibility with earlier versions for existing non-Web-service clients to call a new Web services implementation provided by a SOAP toolkit.



The diagram shows a message set being created from an interface definition (for example, a header file) that is used by an existing client application. A WSDL file is generated from the message set and is used to create a new Web service implementation. A message flow that uses the message set is created to call the

new Web service. The message flow and message set are deployed to a broker, providing the original application interface to the new Web service implementation.

Key to symbols:



Possible uses

You want to migrate an application to a Web service implementation, for example an EJB implementation hosted by an application server to offer better scalability.

However, a significant number of your users have existing clients that cannot be immediately replaced. Existing clients can use the broker to use the new Web service implementation.

Design steps

1. Create a message set for the business messages, for example, by importing an existing interface definition such as a C header file or COBOL copybook.
2. Generate a WSDL definition from the message set.
3. Use a SOAP toolkit or application server to create a suitable Web services implementation based on the WSDL.
4. Develop a message flow to mediate between the original existing client and the new Web service.

At run time

Your message flow receives a request from the existing client, converts it into a Web services request and invokes the Web service. The response from the Web service is converted into a form understood by the existing client.

Part 3. Working with files

Working with files

You can use the FileInput node in your message flows to process data from files. You can use the FileOutput node to send data from a message flow into a file.

Using files is one of the most common methods of storing data. You can create message flows to process data in files, accepting data in files as input message data, and producing output message data for file-based destinations. The following file nodes are provided:

- FileInput node. Use this node to receive messages from files in the broker server's file system or, using FTP or SFTP, in a remote file system. The node generates output message data that any of the output nodes can use, which means that messages can be generated for clients using any of the supported transport protocols to connect to the broker. For more information, see "FileInput node" on page 946.
- FileOutput node. Use this node to write messages to a file in the broker's file system or, using FTP or SFTP, in a remote file system. The node can create new files and replace existing files. For more information, see "FileOutput node" on page 959.

Using these nodes, you can also process large files without having to hold the complete message in memory, and you can simplify the processing of files that have large numbers of repeating entries.

If you want to work with files, read these topics:

- "How the broker processes files"
- "How multiple file nodes share access to files in the same directory" on page 848
- "Using local environment variables with file nodes" on page 849
- "File name patterns" on page 851
- "Archiving" on page 854
- "Reading a file" on page 854
- "Writing a file" on page 862
- "Transferring files securely by using SFTP" on page 870

How the broker processes files

The broker reads files with the FileInput node and writes files with the FileOutput node.

WebSphere Message Broker can read messages from files and write messages to files in the local file system, or on a network file system that appears local to the broker. The following nodes provide this capability:

- FileInput node
- FileOutput node

A file, or a record within a file, is analogous to a message in a queue. The directory that contains the file is analogous to a message queue.

How the broker reads a file

The FileInput node processes messages that are read from files. It searches a specified directory for files that match specified criteria. This input directory is in the file system that is attached to the broker. Optionally, files from a remote FTP or SFTP server can be moved to the local directory whenever the directory is to be scanned. You can find the file that you require by specifying an explicit file name or a file name pattern that includes wildcard characters. If the file is locked, it is ignored at this directory scan.

The FileInput node creates an `mqsitransitin` subdirectory in the input directory. The `mqsitransitin` subdirectory holds and locks the input files while they are being processed. The broker reads the file and propagates a message, or messages, using the contents of the file.

If an execution group that processes files in this input directory is removed, check the `mqsitransitin` subdirectory for partially processed or unprocessed files. Move any such files back into the input directory (and remove the execution group UUID prefix from the file names) so that they can be processed by a different execution group. For more information about the `mqsitransitin` subdirectory, see "How multiple file nodes share access to files in the same directory" on page 848.

You can specify, using the FileInput node, how the records are derived from the file. The contents of a file can be interpreted as:

- A single record ('whole file')
- Separate records, each of a fixed length ('fixed length records')
- Separate records each delimited by a specified delimiter ('delimited records')
- Separate records that are recognized by a parser that you specify ('parser record sequence')

After the file has been successfully processed, it is either deleted from the file system, or moved to an archive subdirectory of the specified (local) directory.

After the last record of the file has been processed successfully, if the End of Data terminal is attached, a further End of Data message is propagated to the End of Data terminal. The End of Data message consists of an empty BLOB message and a `LocalEnvironment.File` structure, and allows explicit end-of-flow processing to be performed in another part of the flow.

The message, or messages, propagated from the file can be used as input to any message flow and output node. You can create a message flow that receives messages from files and generates messages for clients that use any of the supported transports to connect to the broker.

Whenever a message is propagated from a file, the FileInput node stores certain information about the file in the LocalEnvironment.File message tree. This includes the offset of the record of the message in the file being processed and the record number in that file. In addition, when wildcards are used in a file name pattern, the characters matched in the file name are placed in the WildcardMatch element of the local environment tree.

How the broker writes a file

The FileOutput node writes messages to files in the broker's file system. When a message is received on the node's In terminal, it creates and writes a file as a series

of one or more records. One record is written to a file for every message received. The name of the file is specified either by a file name pattern in the node or an explicit file name that is either specified in the node or derived from the message.

You can specify how the records are accumulated in files:

- A single record ('whole file'). The file that is created consists of one record.
- Concatenated records ('unmodified records'). The records are not padded to any required length, and they are not separated by a delimiter.
- Uniform length records ('fixed-length records'). Records that are shorter than the specified length are padded to the required length.
- Separate records ('delimited records'). Records are either terminated or separated by a specified delimiter.

The message flow informs the output node that there are no more records to write by sending a message to its Finish File terminal. The file is then moved to the specified output directory. Optionally, the file can be moved to a directory on a remote FTP or SFTP server identified by properties of the node.

If the node is producing a single record from the file ('whole file'), the file is moved immediately to the output directory without requiring a message to be propagated to the Finish File terminal. In this case, any message sent to the node's Finish File terminal has no effect on any file, but the message is still propagated to a flow attached to the node's End of Data terminal.

Recognizing file records as messages to be parsed

Use the FileInput node to segment your input file into messages that are to be parsed. You use the MRM or XMLNSC parsers.

The node segments your input file into messages that are to be parsed by one of the following parsers:

- MRM Custom Wire Format (CWF)
- MRM Tagged Delimited String Format (TDS)
- XMLNSC

The Message domain property of the node specifies the parser to use; MRM or XMLNSC. Specify Parsed Record Sequence for the Record detection property so that the node splits the file into messages to be parsed by either the MRM parser or XMLNSC parser.

The MRM parser

If you select an MRM parser, ensure that the message model has a defined message boundary and does not rely on the end of the bitstream to stop the parse. If the final element has a *maxOccurs* value of -1, the parser continues to read bytes until the end of the bitstream or until it encounters bytes that cause a parsing exception. In either case, the parser is unable to identify the end of one message and the start of the next. If you use *Data Element Separation* = `Use Data Pattern`, ensure that the pattern recognizes a specified number of bytes. Be aware, therefore, that a pattern of `*` identifies all available characters and so would read an entire input file.

If you use delimited separations with message group indicators and terminators, ensure that the combination of group indicator and terminator does not match a record delimiter. For example, a message might start with a left brace ({) and end

with a right brace (}). If there is a delimiter of }{ within the message, this matches the boundary between multiple messages; as a result, a delimiter within the current message might be identified as a message boundary. This might cause bytes in a subsequent message to be included in the current message causing parser exceptions or unexpected content in the parse tree.

The XMLNSC parser

If you select the XMLNSC parser, the end of the root tag marks the end of the message. XML comments, XML processing instructions and white space that appear after the end of the XML message are discarded. The start of the next XML message is marked either by the next XML root tag or the next XML prolog.

How multiple file nodes share access to files in the same directory

When a message flow uses the FileInput or FileOutput node, additional instances might be associated with the message flow, or file nodes in other message flows in the same execution group (or other execution groups) might refer to files in the same directory. WebSphere Message Broker controls the way in which multiple processes read from and write to files by moving the files to the `mqsitransitin` directory during processing, and locking them while they are being processed. The `mqsitransitin` directory is a subdirectory of the input directory specified in the FileInput node.

The broker locks the files that are being read by the FileInput node or written by the FileOutput node. This prevents other execution groups from reading or changing the files while they are being processed. The broker unlocks the file:

- When a FileInput node finishes processing the input file
- When a FileOutput node finishes writing the file and moves it from the transit directory to the output directory

Reading a file

When the FileInput node reads a file, it first moves the file into the `mqsitransitin` directory, where it is held during processing. A prefix (containing the UUID of the execution group) is added to the file name to indicate which execution group is processing the file. While the file is in this directory, no other execution groups can access the file. The broker maintains a `lock` subdirectory in the `mqsitransitin` directory, to ensure that files in the input directory are accessed by only one execution group at a time.

If multiple message flows or instances within an execution group are reading from the same input directory, only one instance (of one message flow) is allocated to reading it. Each record in the file is serially processed by this instance. Other instances of the message flow, or other message flows, can simultaneously process other files, the names of which match the pattern specified in the node's File name or pattern property.

While a file is being processed, the file system is used to lock the file. This prevents other programs, including other execution groups, from reading, writing, or deleting the file while it is being processed by the file nodes.

While a FileInput node is reading a file, the file remains in the `mqsitransitin` directory until it has been fully processed (or until an unrecoverable error occurs). If the file is to be retained, it is held in a subdirectory.

When the file has been processed, it is moved from the mqsitransitin directory back to the input directory. However, if the execution group stops unexpectedly while the file is in the mqsitransitin directory, you can manually restore the input file to the input directory by removing the execution group UUID prefix from the file name, and then moving it to the input directory. The input file is then processed by the next FileInput node that scans the directory.

Writing a file

Files that are created and written by a FileOutput node are put in the output directory when they are finished. While records are being added to a file, it is kept in the mqsitransit subdirectory.

Each record is written by a single message flow instance. All message flow instances that are configured to write records to a specific file can append records to that file. Because instances can run in any order, records that they write might be interleaved, which means that the sequence of records might be altered. If you need to maintain the sequence of records in the output file, ensure that only one FileOutput node instance uses the file. You can do this by configuring the message flow that contains the node to use the node's additional instances pool with zero instances, and by ensuring that other message flows do not write to the same file.

While a file is being processed, the file system is used to lock the file. This prevents other programs, including other execution groups, from reading, writing, or deleting the file while it being processed by the file nodes. This lock is retained for a short period after a FileOutput node writes to the file without finishing it, leaving it in the transit directory. If message flows that are in the same execution group use the same output file and run sufficiently quickly, the broker does not need to relinquish the lock before the file is finished. However, if the message flows have longer intervals between execution, the broker relinquishes the lock and another process or execution group can acquire a lock on the file. If you need to prevent this behavior, do not share output directories across execution groups.

Using local environment variables with file nodes

You can use fields in the local environment to dynamically alter the behavior of the FileInput and FileOutput nodes. You can also find what values the output node used to process the file.

These fields are available in the following message tree structures:

- LocalEnvironment.File
- LocalEnvironment.WrittenDestination.File
- LocalEnvironment.Destination.File
- LocalEnvironment.Wildcard.WildcardMatch

LocalEnvironment.File fields

When you use the FileInput node, it stores information that you can access in the LocalEnvironment.File message tree. The fields in this structure are described in the following table:

Element Name	Element Data Type	Description
Directory	CHARACTER	Absolute directory path of the input directory in the form used by the file system of the broker. For example, on Windows systems, this starts with the drive letter prefix (such as C:).

Element Name	Element Data Type	Description
Name	CHARACTER	File name and extension.
LastModified	TIMESTAMP	Date and time the file was last modified.
TimeStamp	CHARACTER	Date and time the input node started processing the file in the Coordinated Universal Time (UTC) zone, as a character string. This data is the string used to create archive and backout file names if a timestamp is included.
The following elements contain data about the current record:		
Offset	INTEGER	Start of the record within the file. The first record starts at offset 0. If this element is part of the End of Data message tree, this value is the length of the input file.
Record	INTEGER	Number of the record within the file. The first record is record number 1. If this element is part of the End of Data message tree, this value is the number of records.
Delimiter	CHARACTER	The characters used to separate this record from the preceding record, if Delimited is specified in Record detection. The first record has a null delimiter. If this element is part of the End of Data message tree, this value is the delimiter that follows the last record, if any.
IsEmpty	BOOLEAN	Whether the record propagated by the message flow is empty. It is set to TRUE if the current record is empty. If this element is part of the End of Data message tree, this value is always set to TRUE.

This structure is propagated with each message written to the Out terminal of the FileInput node and with the empty message written to the End of data terminal.

LocalEnvironment.WrittenDestination.File fields

When you use the FileOutput node, it stores information that you can access in the LocalEnvironment.WrittenDestination.File message tree. The fields in this structure are described in the following table:

Element Name	Element Data Type	Description
Directory	CHARACTER	Absolute directory path of the output directory in the form used by the file system of the broker. For example, on Windows systems, this starts with the drive letter prefix (such as `C:`).
Name	CHARACTER	File name of the output file.
Action	CHARACTER	Possible values are: • `Replace` if an output file of the same name is replaced. • `Create` if a new output file is created. • `Append` if this is associated with a record that is appended to an output file. • `Finish` if a Finish File message is received and no file is found to finish (for example, if Record is Whole File is specified and a message is sent to the Finish File terminal). • `Transmit` if the file was transferred by FTP or SFTP and the file was not retained.
Timestamp	CHARACTER	The date and time, in character string form, when the node started to process this file. This is the value which prefixes the names of files that are archived if you specify Time Stamp, Archive and Replace Existing File in the Output file action property on the **Basic** tab.

LocalEnvironment.Destination.File fields

When you use the FileOutput node, you can override its directory and name properties with elements in the message tree. The default location for these overrides is LocalEnvironment.Destination.File, although you can change this location by using the properties on the Request directory property location and Request file name property location on the FileOutput node. The fields of this structure are described in the following table:

Element Name	Element Data Type	Description
Directory	CHARACTER	Absolute or relative directory path of the output directory in the form that is used by the file system of the broker. For example, on Windows systems, this path starts with the drive letter prefix (such as C:) and use a backslash (\) as the directory delimiter. On UNIX systems, the path includes a slash (/) as the directory delimiter.
Name	CHARACTER	File name of the output file. The FileOutput node does not perform wildcard replacement on this element's value. For example, if its value is `Input*.txt`, the FileOutput node tries to write to a file with an asterisk (*) in its name. It might or might not succeed, depending on whether an asterisk is a valid character for files in the file system to which it is writing.

LocalEnvironment.Wildcard.WildcardMatch field

On the FileInput node, you can specify a file name pattern that contains wildcard characters. The FileInput node copies the characters in the file name matched by wildcards, together with any intermediate characters, to LocalEnvironment.Wildcard.WildcardMatch.

Element Name	Element Data Type	Description
WildcardMatch	CHARACTER	The character string in the file name matched by wildcards in the file name pattern.

On the FileOutput node, you can use a wildcard character in the file name pattern. If you include the single wildcard character, '*', in the file name pattern, the node uses the value that is stored in LocalEnvironment.Wildcard.WildcardMatch. This is useful if you have a message flow where the FileInput and FileOutput nodes are working with the same file; you can preserve the name of the input file on the FileOutput node. You can also use standard methods for manipulating the value of the WildcardMatch element to whatever you want; you do not have to use a FileInput node.

See “File name patterns” for more information.

File name patterns

You can specify a file name pattern, using wildcard characters, to identify a file to be read by the FileInput node. You can also specify a file name pattern, using a single wildcard character, to name the file to be created by the FileOutput node.

Using file name patterns with the FileInput node

The FileInput node reads files from a specified directory and propagates messages based on the contents of these files. Only files with names that match a pattern

(the input pattern), as specified in the FileInput node's File name or pattern property, are read. The match might be with a file name or a character sequence (a pattern). A pattern is a sequence containing at least one of the following wildcard characters:

Wildcard character	Description	Example
*	Any sequence of zero or more characters	`*.xml` matches all file names with an `xml` extension
?	Any single character	`f??????.csv` matches all file names consisting of the letter `f` followed by six characters, then the sequence `.csv`.

The default pattern is *, which matches all file names.

You cannot specify file names that contain the following characters: the asterisk ('*'), the question mark ('?'), or file name separator characters ('/' and' \').

For example, if you want the FileInput node to process all files that have a certain extension, such as `xml`, set its File name or pattern property to *.xml and the node will process all files in the directory that have this extension.

If you deploy the flow to a Windows server, file names match the pattern irrespective of case. However, if you deploy the flow to a Linux, UNIX, or z/OS server, file names must match the pattern character string and its case.

Pattern matching

The FileInput node sets the LocalEnvironment.Wildcard.WildcardMatch element to the string matched by wildcards in the file name. The following are some examples of pattern matching with the value in this element, where the value in the FileInput node's File name or pattern property is File????.from*.xml:

- If the FileInput node finds a file with the file name `File1234.fromHQ.xml`, there is a match. The value in the LocalEnvironment.Wildcard.WildcardMatch element is set to `1234.fromHQ` and the node processes the file.
- If the file name is `File123.fromHQ.xml`, there is no match because there are insufficient characters between the `File` and `.from` elements of the file name. The FileInput node ignores this file.
- If the file name is `File2345.from.xml`, there is a match. The value in the LocalEnvironment.Wildcard.WildcardMatch element is set to `2345.from` and the node processes the file. In this example, the * in the character string in the File name or pattern property matches a string of zero characters. If you require the character string between the `from` and `.xml` elements of the file name to always have at least one character, you specify the File name or pattern property with a value of File????.from?*.xml.

Using file name patterns with the FileOutput node

The node writes messages to files that it creates or replaces in the broker's file system. Only patterns containing a single wildcard character (the asterisk, '*') are allowed in this property. The file name to be used is determined in the following way:

- If the file name property contains no wildcard, the value of this property is the name of the file created. This value must be a valid file name on the file system that hosts the broker to which the message flow is deployed.

- If the file name property contains a single wildcard, the value of the element LocalEnvironment.Wildcard.WildcardMatch in the current message replaces the wildcard character, and the resulting value is the name of the file created. This value must be a valid file name on the file system that hosts the broker to which the message flow is deployed. If the WildcardMatch value is not found, the wildcard character is replaced by the empty string.

You cannot specify file names that contain the following characters: the asterisk ('*'), the question mark ('?'), file name separator characters ('/' and' \'). The name of the file can be overridden by values in the current message.

If the File name or pattern property on the FileOutput node is empty, the name must be overridden by the current message. Wildcard substitution occurs only if this property is not overridden in this way.

File names are passed to the file system to which the broker has access, and have to respect the conventions of these file systems. For example, file names on Windows systems are not case-sensitive; on UNIX systems, file names that differ by case are considered distinct.

Example: If the FileInput node has `*.out` in the File name or pattern property, and the incoming file is `myfile`, the name of the outgoing file is `myfile.out`.

FTP and SFTP considerations

You can use the FileInput node to transfer files from a remote FTP or SFTP server and process them. Only files with names that match the file name pattern specified in the node are read. If your broker is on an operating system that respects case sensitivity (such as UNIX), you might specify a pattern that includes a combination of uppercase and lowercase characters. If you then use this pattern to process files that are in a directory on a remote FTP or SFTP server, and this server is running on an operating system that does not respect case sensitivity (such as Windows), file name matching might fail with the result that no files are processed. This failure occurs because the file names on the remote server are not in mixed case. If your broker is on an operating system that does not respect case sensitivity, any pattern that you specify might be matched by more than one file on a remote FTP or SFTP server that is running on an operating system on which case sensitivity is significant. Each of these files is then processed sequentially.

You can use the FileOutput node to write files to a remote FTP or SFTP server. Only files with names that match the pattern specified in the node are written. If your broker is running on an operating system that respects case sensitivity (such as UNIX), you might specify a pattern that includes a combination of uppercase and lowercase characters. However, if you then use this pattern to write files to a directory on a remote FTP or SFTP server running on an operating system that does not respect case sensitivity (such as Windows), the file name is written in uppercase rather than in the way that it was specified in your pattern.

If the name of a file on a remote FTP server contains one or more characters that are not valid on the operating system on which the broker where you specified the file name pattern is running, the file is not transferred from the FTP server for processing by the FileInput node.

Archiving

Files that are successfully processed by the FileInput node or FileOutput node can optionally be moved to the `mqsiarchive` subdirectory of the input or output directory.

The input directory of the FileInput node has a subdirectory called `mqsiarchive`. The output directory of the FileOutput node also has a subdirectory called `mqsiarchive`.

FileInput node

Files that are processed successfully by the FileInput node are moved to the `mqsiarchive` subdirectory if the FileInput node's Action on successful processing property is set to Move to Archive Subdirectory or Add Timestamp and Move to Archive Subdirectory.

Select the Replace duplicate archive files check box to overwrite existing files in the `mqsiarchive` subdirectory. If you do not set this option, and a file with the same name already exists in the archive subdirectory, the node stops processing files. Every time that the node returns from its polling wait period, it issues a pair of messages, BIP3331 and a more specific one describing the problem. To avoid writing too many messages, duplicate messages are suppressed for increasing periods of time, until eventually they are issued only about once every hour. In this circumstance, the system administrator must stop the flow, correct the problem, then restart the flow.

Clear the Replace duplicate archive files check box only if you are sure either that the input files have unique names, or that some other process will remove a file from the archive directory before the FileInput node processes another of the same name. If you cannot ensure this, either specify Add Timestamp and Move to Archive Subdirectory in the Action on successful processing property so that archived files have unique names, or select the Replace duplicate archive files check box

FileOutput node

Files that are processed successfully by the FileOutput node are moved to the `mqsiarchive` subdirectory if the FileOutput node's Output file action property is set to Archive and Replace Existing File or Time Stamp, Archive and Replace Existing File.

Select the Replace duplicate archive files check box to overwrite existing files in the `mqsiarchive` subdirectory. If you do not set this option, and a file with the same name already exists in the archive subdirectory, the node generates an exception when it tries to move the successfully processed file and the file remains in the transit subdirectory.

Reading a file

Use the FileInput node to read files.

This section contains the following topics:

- "Setting the Record detection property of the FileInput node" on page 859

Reading a file on your local file system

Learn how to use the FileInput node to read a file on your local file system and then propagate messages that are based on the contents of that file.

This example shows how one combination of values in the Record detection, Delimiter, and Delimiter type properties can be used to extract messages from a file. The example describes the FileInput node of a message flow and assumes that the rest of the flow has already been developed. It is also assumed that a Windows system is being used. To complete this example task, you must first have added a FileInput node to a message flow. You also need the following resources:

- An input file. To follow this example scenario, create an input file called `test_input1.xml` with the following content:

  ```
  <Message>test1</Message>
  <Message>testtwo</Message>
  <Message>testthree</Message>
  ```

  Each line ends with a line terminator; on a Windows system, this comprises carriage return and line feed characters (X'0D0A'). Put this file into directory `C:\FileInput\TestDir`.
- A message set. This example uses a message set called `xml1` which uses the XMLNSC parser. Message set `xml1` models messages of the following form:

  ```
  <Message>...</Message>
  ```

Complete the following steps:

1. Set the required node properties on the FileInput node. The following table summarizes the FileInput node properties that you should set, which tab they appear on and the value that you should set in order to follow this example:

Tab	Property	Value
Basic	Input directory	C:\FileInput\TestDir
	File name or pattern	test_input1.xml
	Action on successful processing	Move to Archive Subdirectory
	Replace duplicate archive files	Selected
Input Message Parsing	Message domain	XMLNSC
	Message set	xml1
Polling	Polling interval	3
Retry	Action on failing file	Add Time Stamp and Move to Backout Subdirectory
Records and Elements	Record detection	Delimited
	Delimiter	DOS or UNIX Line End
	Delimiter type	Postfix
FTP	FTP	Not selected

2. Deploy the message flow to the broker. See Deploying.

The following actions occur when you perform these steps:

1. The file is processed. In accordance with the values set in the properties on the **Records and Elements** tab, the FileInput node detects records that are separated by DOS or UNIX end-of-line characters and creates a message for each one that it finds. It propagates three messages to the flow attached to the Out terminal:
   - Message 1:
     ```
     <Message>test1</Message>
     ```
   - Message 2:
     ```
     <Message>testtwo</Message>
     ```
   - Message 3:
     ```
     <Message>testthree</Message>
     ```
2. If a flow is attached to the End of Data terminal, the End of Data message is propagated after the last record in the file has been processed.
3. When processing is complete, the file `test_input1.xml` is moved to the `mqsiarchive` subdirectory, `C:\FileInput\TestDir\mqsiarchive\test_input1.xml`. If a file called `test_input1.xml` already exists in the `mqsiarchive` subdirectory, it is overwritten.
4. If the message flow fails, retry processing is attempted according to the values set in the properties of the FileInput node. In this example task, a time stamp is added to the file name and the file is moved to the `mqsibackout` directory. Here is an example of the path to such a file: `C:\FileInput\TestDir\mqsibackout\20070928_150234_171021_test_input1.xml`.

To see the effects of specifying other combinations of values in the Record detection, Delimiter, and Delimiter type properties of the FileInput node, see "Setting the Record detection property of the FileInput node" on page 859.

The following samples also show how to use this node:

- Batch Processing
- WildcardMatch

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

Reading a file on a remote FTP or SFTP directory

Use a FileInput node to read a file in a directory on a remote FTP or SFTP server and then propagate messages that are based on the contents of that file.

Before you start:

This example is an extension of the example described in "Reading a file on your local file system" on page 855 and it describes how to use a FileInput node in a message flow. The instructions assume that you are using a Windows operating system and that you have created a message flow containing a FileInput node. You also require the following resources:

- An FTP or SFTP server. Ensure that an FTP or SFTP server exists, with the following settings:

  Server *ftpserver.hursley.abc.com*

  Port *21* (for FTP) or *22* (for SFTP)

  Working directory
  /ftpfileinput

Userid

myuserid

Password

mypassword

These values are for the purposes of this example only. If you use other values, record them so that you can set the appropriate values during the task.

- A security identity. Use the mqsisetdbparms command to define a security identity called *myidentity* for your user and password details.

  If you want to connect to an FTP server, the security identity must have an `ftp::` prefix, to enable the file nodes to find the identity definition. For example, use the following command for a broker called *MyBroker*:

  ```
  mqsisetdbparms MyBroker -n ftp::myidentity -u myuserid -p mypassword
  ```

  If you want to connect to an SFTP server, the security identity must have an `sftp::` prefix, as shown in the following example:

  ```
  mqsisetdbparms MyBroker -n sftp::myidentity -u myuserid -p mypassword
  ```

  You can also configure a connection to an SFTP server to use public key authentication, by specifying an SSH identity file and pass phrase, instead of a password. For example:

  ```
  mqsisetdbparms MyBroker -n sftp::myidentity -u myuserid -i identity_file -r passphrase
  ```

  For more information about configuring connections to an SFTP server, see "Transferring files securely by using SFTP" on page 870.

- An input file. To follow this example scenario, create an input file called `test_input1.xml` with the following content:

  ```
  <Message>test1</Message>
  <Message>testtwo</Message>
  <Message>testthree</Message>
  ```

  Each line ends with a line terminator that is suitable for the system on which the FTP or SFTP server is found. Do not put this file in the input directory but, instead, put it in the FTP or SFTP server directory `/ftpfileinput`.

- A message set. This example uses a message set called *xml1*, which uses the XMLNSC parser. Message set *xml1* models messages of the following form:

  ```
  <Message>...</Message>
  ```

Complete the following steps:

1. Set the required node properties on the FileInput node. The following table summarizes the FileInput node properties that you must set, the tab on which they are displayed, whether they are mandatory, and the required values.

Tab	Property	Value
Basic	Input directory	C:\FileInput\TestDir If the input directory does not exist, no files are processed, even if you are processing files over FTP or SFTP.
	File name or pattern	test_input1.xml
	Action on successful processing	Move to Archive Subdirectory
	Replace duplicate archive files	Selected
Input Message Parsing	Message domain	XMLNSC
	Message set	xml1

Tab	Property	Value
Polling	Polling interval	3
Retry	Action on failing file	Add Time Stamp and Move to Backout Subdirectory
Records and Elements	Record detection	Delimited
	Delimiter	DOS or UNIX Line End
	Delimiter type	Postfix
FTP	Remote transfer	Selected
	Transfer protocol	FTP or SFTP
	Remote server and port	ftpserver.hursley.abc.com
	Security identity	myidentity
	Server directory	/ftpfileinput
	Transfer mode	ASCII (for FTP only)
	Scan delay	45

If you used other values for your FTP or SFTP server resource, enter those values. The settings used here are identical to those used in the example in "Reading a file on your local file system" on page 855, except that the Remote transfer property has been selected and there are now properties on the **FTP** tab. If you clear the Remote transfer property, the node operates as it does in the example in "Reading a file on your local file system" on page 855; the properties on the **FTP** tab remain set but are ignored.

2. Deploy the message flow to the broker. See Deploying.

The following actions occur when you perform these steps:

1. The file `test_input1.xml` is transferred from the FTP or SFTP server directory (`/ftpfileinput`) to the local directory (`C:\FileInput\TestDir`). The file is deleted from the FTP or SFTP server directory.
2. The FileInput node detects records that end with a DOS or UNIX line end and creates a message for each one that it finds, as defined by the properties on the **Records and elements** tab. The node propagates three messages to the message flow that is attached to the Out terminal:
   - Message 1:
     ```
     <Message>test1</Message>
     ```
   - Message 2:
     ```
     <Message>testtwo</Message>
     ```
   - Message 3:
     ```
     <Message>testthree</Message>
     ```
3. If a node is attached to the End of Data terminal, the End of Data message is propagated after the last record in the file has been processed.
4. When processing is complete, the file `test_input1.xml` is moved to the `mqsiarchive` subdirectory `C:\FileInput\TestDir\mqsiarchive`. If a file called `test_input1.xml` exists in the `mqsiarchive` subdirectory, it is overwritten.
5. If the message flow fails, retry processing is attempted according to the values set in the properties of the FileInput node. In this example task, a time stamp is added to the file name and the file is moved to the `mqsibackout` directory. Here is an example of the path to such a file: `C:\FileInput\TestDir\mqsibackout\20070928_150234_171021_test_input1.xml`.

If an error occurs on the FTP side, stating that access is denied, a 0–byte file is created and moved to the mqsibackout directory. A 0–byte file is created in the mqsibackout directory for every FTP attempt that fails.

Because the Remote transfer property is selected, the FTP scan delay of 45 seconds overrides the polling interval of 3 seconds.

For more information, see "Setting the Record detection property of the FileInput node," which shows the effects of specifying other combinations of values in the Record detection, Delimiter, and Delimiter type properties of the FileInput node.

The following samples also show how to use this node:
- Batch Processing
- WildcardMatch

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

Setting the Record detection property of the FileInput node

Set the Record detection and other properties on the FileInput node's **Records and Elements** tab to read files in different formats.

The following examples are based on those described in "Reading a file on your local file system" on page 855 and "Reading a file on a remote FTP or SFTP directory" on page 856. In each case, the input file to use, the property settings, and the expected results are described.

Example 1. Records are separated by a DOS or UNIX line end

This example is identical to the one described in "Reading a file on your local file system" on page 855 or "Reading a file on a remote FTP or SFTP directory" on page 856. Create an input file called `test_input1.xml` with the following content:

```
<Message>test1</Message>
<Message>testtwo</Message>
<Message>testthree</Message>
```

Each line ends with a line terminator.

The properties to set are:

Tab	Property	Value
Records and Elements	Record detection	Delimited
	Delimiter	DOS or UNIX Line End
	Delimiter type	Postfix

The FileInput node detects records that end with a DOS or UNIX line end and creates a message for each one that it finds.

The result is the propagation of three messages, as follows:
- Message 1:

  ```
  <Message>test1</Message>
  ```
- Message 2:

<Message>testtwo</Message>

- Message 3:

<Message>testthree</Message>

The DOS or UNIX line end is not part of any propagated message.

Example 2. Records are separated by a custom delimiter

Create an input file called test_input2.xml with the following content:

<Message>test01</Message>,<Message>test001</Message>,<Message>test0001</Message>

There should be no line terminator at the end of this file data; the XMLNSC parser ignores the line terminator if it is present.

In addition to the property settings described in "Reading a file on your local file system" on page 855 or "Reading a file on a remote FTP or SFTP directory" on page 856, set these properties:

Tab	Property	Value
Basic	File name or pattern	test_input2.xml
Records and Elements	Record detection	Delimited
	Delimiter	Custom Delimiter
	Custom delimiter	2C
	Delimiter type	Infix

The hexadecimal X'2C' represents a comma in ASCII. On other systems, you might need to use a different hexadecimal code.

The FileInput node detects the comma character and uses it to separate records. Because the value of the Delimiter type property is Infix, a comma is not required at the end of the file.

The result is the propagation of three messages, as follows:

- Message 1:

<Message>test01</Message>

- Message 2:

<Message>test001</Message>

- Message 3:

<Message>test0001</Message>

The comma character is not part of any propagated message. There are no commas in the bodies of the message in this example; if the message bodies did contain commas, the records would be split at those points resulting in incorrectly formed messages being propagated to the rest of the flow.

Example 3. Records are separated by a fixed number of bytes

Create an input file called test_input3.xml with the following content:

<Message>123456789</Message><Message>abcdefghi</Message><Message>rstuvwxyz</Message>

There should be no line terminator at the end of this file.

In addition to the property settings described in “Reading a file on your local file system” on page 855 or “Reading a file on a remote FTP or SFTP directory” on page 856, set these properties:

Tab	Property	Value
Basic	File name or pattern	test_input3.xml
Records and Elements	Record detection	Fixed Length
	Length	28

The FileInput node splits the input file into records each 28 bytes long.

The result is the propagation of three messages, as follows:

- Message 1:

  ```
  <Message>123456789</Message>
  ```

- Message 2:

  ```
  <Message>abcdefghi</Message>
  ```

- Message 3:

  ```
  <Message>rstuvwxyz</Message>
  ```

Each message is 28 bytes long. If the file contains trailing bytes, for example a carriage return-line feed pair, a further message containing these bytes is propagated; this might or might not be recognized by the message domain, message set and message type assigned to parse the message.

Example 4. Records are whole files

Create an input file called `test_input4.xml` with the following content:

```
<Message>Text string of a length decided by you, even including line
terminators, as long as it only contains this tag at the end.</Message>
```

There should be no line terminator at the end of this file; if there is one, it has no effect.

In addition to the property settings described in “Reading a file on your local file system” on page 855 or “Reading a file on a remote FTP or SFTP directory” on page 856, set these properties:

Tab	Property	Value
Basic	File name or pattern	test_input4.xml
Records and Elements	Record detection	Whole File

The FileInput node does not split the file; it propagates all of the file's content as a single record to be parsed by the message domain, message set and message type as specified on the node. In this example, you are using the XMLNSC parser and message set `xml1`, so the message should be recognized.

The result is the propagation of one message, as follows:

- Message 1:

```
<Message>Text string of a length decided by you, even including line terminators, as long as
it only contains this tag at the end.</Message>
```

Trailing bytes (for example, line terminators) are included in this.

Example 5. Records are recognized as separate messages by the parser specified in the Message domain property

Create an input file called `test_input5.xml` with the following content:

```
<Message>Text string of a length decided by you  </Message><Message>and another</Message>
<Message>and another on a new line</Message>
```

Line terminators at the end of this file, or at the end of lines, are acceptable.

In addition to the property settings described in "Reading a file on your local file system" on page 855 or "Reading a file on a remote FTP or SFTP directory" on page 856, set these properties:

Table 13.

Tab	Property	Value
Basic	File name or pattern	test_input5.xml
Records and Elements	Record detection	Parsed Record Sequence

The FileInput node defers to the parser to determine the record boundaries. In this example, message set `xml1` in domain XMLNSC should recognize the complete `<Message>` XML format. XMLNSC absorbs trailing white space (for example, line terminators).

The result is the propagation of three messages, as follows:

- Message 1:

```
<Message>Text string of a length decided by you  </Message>
```

- Message 2:

```
<Message>and another</Message>
```

- Message 3:

```
<Message>and another on a new line</Message>
```

Trailing white space (for example, line terminators) are included in the messages.

Writing a file

Use the FileOutput node to write files.

This section contains the following topics:

- "Writing a file to your local file system"
- "Writing a file to a remote FTP or SFTP server" on page 864
- "Setting the FileOutput node's Record definition property" on page 867

Writing a file to your local file system

Use a FileOutput node to write a file to a specified directory on your local file system.

This example shows you how one combination of values in the Record definition, Delimiter, and Delimiter type properties result in the creation of a file from multiple messages. The example describes the FileOutput node of a message flow and assumes that the rest of the flow has been developed. It is also assumed that a Windows system is being used. To complete this example task, you must first have

added a FileOutput node to a message flow. You also need to ensure that the following messages are produced by the flow preceding the FileOutput node:

- Three input messages, which are sent, in this order, to the In terminal of the FileOutput node:
  - Message 1:

    ```
    <Message>test1</Message>
    ```
  - Message 2:

    ```
    <Message>testtwo</Message>
    ```
  - Message 3:

    ```
    <Message>testthree</Message>
    ```

  These messages can be produced, for example, by the XMLNSC domain with a message set which recognizes XML with the following form:

  ```
  <Message>...</Message>
  ```
- A final message, which is sent to the Finish File terminal of the FileOutput node after the first three messages have been sent:

  ```
  <thiscanbe>anything</thiscanbe>
  ```

Complete the following steps:

1. Set the required node properties on the FileOutput node. The following table summarizes the FileOutput node properties that you must set, which tab they appear on, and the value that you must set in order to follow this example:

Tab	Property	Value
Basic	Directory	C:\FileOutput\TestDir
	File name or pattern	test_output1.xml
	Output file action	Time Stamp, Archive and Replace Existing File (or Create if File does not Exist)
	Replace duplicate archive files	Selected
Records and Elements	Record definition	Record is Delimited Data
	Delimiter	Broker System Line End
	Delimiter type	Postfix
FTP	FTP	Cleared

2. Deploy the message flow to the broker. See Deploying.
3. Send the first three messages to the In terminal of the FileOutput node.
4. Send the final message to the Finish File terminal of the FileOutput node.

The following actions occur when you perform these steps:

1. The file is processed. In accordance with the values set in the properties of the FileOutput node, the node generates one record per message with a local file system line terminator after each one. The file contains the following data, each line terminated by a carriage return (X'0D') and line feed (X'0A') pair of characters (on a Windows system):

   ```
   <Message>test1</Message>
   <Message>testtwo</Message>
   <Message>testthree</Message>
   ```

2. Records are accumulated in file `test_output1.xml` in the `C:\FileOutput\TestDir\mqsitransit` directory. When the final message is sent to the Finish File terminal, the file is moved to the output directory, `C:\FileOutput\TestDir` directory.
3. If a file of the same name already exists in the output directory, the existing file is renamed and moved to the `mqsiarchive` directory. For example, this might result in the creation of file:

   ```
   C:\FileOutput\TestDir\mqsiarchive\20081124_155346_312030_test_output1.xml
   ```

   If a file of this name already exists in this archive directory, it is overwritten in accordance with the Replace duplicate archive files property selected on the FileOutput node.

See "Setting the FileOutput node's Record definition property" on page 867 to see the results of running this task with different values set in the Record definition, Delimiter, and Delimiter type properties of the FileOutput node.

The following samples also show how to use this node:

- File Output
- Batch Processing
- WildcardMatch

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

Writing a file to a remote FTP or SFTP server

Use a FileOutput node to write a file to a directory on a remote FTP or SFTP server.

This example shows you how one combination of values in the Record definition, Delimiter, and Delimiter type properties result in the creation of a file from multiple messages. The example is an extension of the example described in "Writing a file to your local file system" on page 862, and describes the use of a FileOutput node in a message flow.

These instructions assume that you are using a Windows system, and that you have already created a message flow containing a FileOutput node. You also require the following resources:

- An FTP or SFTP server. Ensure that an FTP or SFTP server exists with the following settings, so that you can follow this example scenario:

  Server *ftpserver.hursley.abc.com*

  Port *21* (for FTP) or *22* (for SFTP)

  Working directory
  /ftpfileoutput

  Userid
  myuserid

  Password
  mypassword

These values are for the purposes of this example only. If you use other values, record them so that you can set the appropriate values when you follow the instructions in this task.

- A security identity. Use the mqsisetdbparms command to define a security identity called *myidentity* for your user and password details.

  If you want to connect to an FTP server, the security identity must have an `ftp::` prefix, to enable the file nodes to find the identity definition. For example, use the following command for a broker called *MyBroker*:

  ```
  mqsisetdbparms MyBroker -n ftp::myidentity -u myuserid -p mypassword
  ```

  If you want to connect to an SFTP server, the security identity must have an `sftp::` prefix, as shown in the following example:

  ```
  mqsisetdbparms MyBroker -n sftp::myidentity -u myuserid -p mypassword
  ```

  You can also configure a connection to an SFTP server to use Public Key authentication, by specifying an SSH identity file and pass phrase, instead of a password. For example:

  ```
  mqsisetdbparms MyBroker -n sftp::myidentity -u myuserid -i identity_file -r passphrase
  ```

  For more information about configuring connections to an SFTP server, see "Transferring files securely by using SFTP" on page 870.

- The following messages, which must be produced by the message flow preceding the FileOutput node:
  - Three input messages. These messages are sent, in this order, to the In terminal of the FileOutput node:
    - Message 1:

      ```
      <Message>test1</Message>
      ```

    - Message 2:

      ```
      <Message>testtwo</Message>
      ```

    - Message 3:

      ```
      <Message>testthree</Message>
      ```

    These messages can be produced, for example, by the XMLNSC domain with a message set that recognizes XML, in the following form:

    ```
    <Message>...</Message>
    ```

  - A final message sent to the Finish File terminal of the FileOutput node after the first three messages have been sent:

    ```
    <thiscanbe>anything</thiscanbe>
    ```

Complete the following steps:

1. Set the required node properties on the FileOutput node. The following table summarizes the FileOutput node properties that you must set, the tabs on which they are displayed, and the values that are used in this example:

Tab	Property	Value
Basic	Directory	C:\FileOutput\TestDir
	File name or pattern	test_output1.xml
	Output file action	Time Stamp, Archive and Replace Existing File (or Create if File does not Exist)
	Replace duplicate archive files	Selected

Tab	Property	Value
Records and Elements	Record definition	Record is Delimited Data
	Delimiter	Broker System Line End
	Delimiter type	Postfix
FTP	Remote transfer	Selected
	Transfer protocol	FTP or SFTP
	Remote server and port	ftpserver.hursley.abc.com
	Security identity	myidentity
	Server directory	/ftpfileoutput
	Transfer mode	ASCII (for FTP only)
	Retain local file after transfer	Selected

If you used other values for your FTP or SFTP server resource, use those values. The settings used here are identical to those used in the example in "Writing a file to your local file system" on page 862 except that the Remote transfer property has been selected and there are now properties on the **FTP** tab. If you clear the Remote transfer property, the node operates as it does in the example in "Writing a file to your local file system" on page 862; the properties on the **FTP** tab remain set but are ignored.

2. Deploy the message flow to the broker. See Deploying.
3. Send the first three messages to the In terminal of the FileOutput node.
4. Send the final message to the Finish File terminal of the FileOutput node.

The following actions occur when you perform these steps:

1. The file is processed. The FileOutput node generates one record per message with a local file system line terminator after each one. The file contains the following data, with each line terminated by a carriage return (X'0D') and line feed (X'0A') pair of characters (on a Windows system):

   ```
   <Message>test1</Message>
   <Message>testtwo</Message>
   <Message>testthree</Message>
   ```

2. Records are accumulated in file `test_output1.xml` in the `C:\FileOutput\TestDir\mqsitransit` directory. When the final message is sent to the Finish File terminal, the file is moved to the remote FTP or SFTP server directory (because the Remote transfer property is selected). As a result, the file `/ftpfileoutput/test_output1.xml` is created.
3. If a file with the same name exists in the remote FTP or SFTP server directory, the existing file is overwritten.

   If the remote FTP server is not running on a Windows system and the Transfer mode property is set to ASCII, the character encoding and line terminator characters might be modified after transfer. For example, on a z/OS FTP server, the ASCII text is typically converted to EBCDIC, and the line terminator character pairs are replaced by EBCDIC new line characters (X'15'). Other FTP servers might treat ASCII transfers differently. If you are using SFTP, the Transfer mode property is ignored and files are sent as Binary files.
4. Because the Retain local file after transfer property is selected, the local file is not deleted but is moved from the `mqsitransit` subdirectory to the output directory, `C:\FileOutput\TestDir`. If a file with the same name exists in the output directory, the existing file is renamed and moved to the `mqsiarchive` directory. For example, the following file might be created:

```
C:\FileOutput\TestDir\mqsiarchive\20081124_155346_312030_test_output1.xml
```

However, if a file with this name exists in this archive directory, it is overwritten according to the value of the Replace duplicate archive files property set on the FileOutput node.

For more information, see “Setting the FileOutput node's Record definition property,” which shows the results of running this task with different values set in the Record definition, Delimiter, and Delimiter type properties of the FileOutput node.

The following samples also show how to use this node:

- File Output
- Batch Processing
- WildcardMatch

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

Setting the FileOutput node's Record definition property

Set the properties on the**Records and Elements** tab of the node to write files in different formats.

The following examples are based on those described in “Writing a file to your local file system” on page 862 and “Writing a file to a remote FTP or SFTP server” on page 864. In all examples, it is assumed that the same messages are sent to the FileOutput node; three to the In terminal and one to the Finish File terminal:

- Three input messages which are sent, in this order, to the In terminal of the FileOutput node:
  - Message 1:

    ```
    <Message>test1</Message>
    ```
  - Message 2:

    ```
    <Message>testtwo</Message>
    ```
  - Message 3:

    ```
    <Message>testthree</Message>
    ```

  These messages can be produced, for example, by the XMLNSC domain with a message set which recognizes XML with the following form:

  ```
  <Message>...</Message>
  ```
- A final message which is sent to the Finish File terminal of the FileOutput node after the first three messages have been sent. It does not matter what this message contains.

The following examples describe the contents of the file or files produced; the disposition of the files created is as in the “Writing a file to your local file system” on page 862 and “Writing a file to a remote FTP or SFTP server” on page 864 topics.

Example 1. Records written are separated by a DOS or UNIX line end

This example is identical to the one described in “Writing a file to your local file system” on page 862 or “Writing a file to a remote FTP or SFTP server” on page 864

864. Specify the node's properties as described in “Writing a file to your local file system” on page 862 or “Writing a file to a remote FTP or SFTP server” on page 864.

These properties result in one file being written. The file contains three records each terminated by a local system line terminator. On a Windows system, this is a carriage return (X'0D') line feed (X'0A') pair of characters; on UNIX systems it is X'0A'.

```
<Message>test1</Message>
<Message>testtwo</Message>
<Message>testthree</Message>
```

Example 2. Records written are separated by a custom delimiter

In addition to the property settings described in “Writing a file to your local file system” on page 862 or “Writing a file to a remote FTP or SFTP server” on page 864, set these properties on the **Records and Elements** tab:

Property	Value
Record definition	Record is Delimited Data
Delimiter	Custom Delimiter
Custom delimiter	0D0A
Delimiter type	Postfix

The hexadecimal X'0D0A' represents a carriage return character followed by a line feed character. On a Windows system, this results in a file identical to the one created in Example 1. On other systems, the result might differ from the result in Example 1; Example 1 uses local system line end characters, whereas Example 2 always puts the X'0D0A' sequence at the end of each line.

Example 3. Records written are padded to a fixed length

In addition to the property settings described in “Writing a file to your local file system” on page 862 or “Writing a file to a remote FTP or SFTP server” on page 864, set these properties on the **Records and Elements** tab:

Property	Value
Record definition	Record is Fixed Length Data
Length (bytes)	30
Padding bytes (hexadecimal)	2A

The hexadecimal character X'2A' represents an asterisk character in ASCII.

The length of each incoming message is 24 bytes, 26 bytes, and 28 bytes respectively. The required fixed length of each record is 30 bytes. Each record is therefore padded by an extra 6 bytes, 4 bytes, and 2 bytes respectively, using the hexadecimal character X'2A'.

One file is written. It contains a single line:

```
<Message>test1</Message>******<Message>testtwo</Message>****<Message>testthree</Message>**
```

Example 4. Records written are not separated by delimiters or padding

In addition to the property settings described in "Writing a file to your local file system" on page 862 or "Writing a file to a remote FTP or SFTP server" on page 864, set this property on the **Records and Elements** tab:

Property	Value
Record definition	Record is Unmodified Data

The records are concatenated with no padding or delimiters.

One file is written with the following content:

```
<Message>test1</Message><Message>testtwo</Message><Message>testthree</Message>
```

There are no trailing bytes or line terminators.

Example 5. Records are written as whole files

In addition to the property settings described in "Writing a file to your local file system" on page 862 or "Writing a file to a remote FTP or SFTP server" on page 864, set this property on the **Records and Elements** tab:

Property	Value
Record definition	Record is Whole File

Three files are created, each containing one record:

- File 1:

  ```
  <Message>test1</Message>
  ```

- File 2:

  ```
  <Message>testtwo</Message>
  ```

- File 3:

  ```
  <Message>testthree</Message>
  ```

Each of these files is created with the same name, one by one, in the mqsitransit directory. If you are following the example in "Writing a file to a remote FTP or SFTP server" on page 864, each file is transferred to the remote FTP server. However, because each file overwrites the previous one, only the third file remains when the task is complete.

After optional transfer, if a copy is retained, each file is moved to the output directory, `C:\FileOutput\TestDir`. In accordance with the properties on the FileOutput node as described in "Writing a file to your local file system" on page 862 or "Writing a file to a remote FTP or SFTP server" on page 864, the second file moved displaces the first file from the output directory which is moved to the mqsiarchive subdirectory with a time stamp added to the file name. When the third file is moved to the output directory, it displaces the second file, causing it to be moved to the mqsiarchive subdirectory and renamed. The final result is files similar to these:

```
C:\FileOutput\TestDir\mqsiarchive\20071101_165346_312030_test_output1.xml
C:\FileOutput\TestDir\mqsiarchive\20071101_165347_312030_test_output1.xml
C:\FileOutput\TestDir\test_output1.xml
```

being File 1, File 2, and File 3 respectively. If FTP processing was enabled, File 3 would also be in the remote FTP server directory and called `test_output1.xml`.

Transferring files securely by using SFTP

You can transfer files securely by using the Secure File Transfer Protocol (SFTP), which enables file transfer by using the Secure Shell (SSH) protocol.

Use the properties on the FTP tab of the FileInput and FileOutput nodes to configure the secure transfer of files.

You can also use the FtpServer configurable service to configure other SFTP properties, including the cipher to be used for SFTP communication, compression level, and strict known host checking.

See the following topics for more information about configuring secure file transfer:

- "Known host checking" on page 871
- "Configuring SFTP file transfer"

Configuring SFTP file transfer

Use an FtpServer configurable service to specify the SFTP settings for a message flow, and to override the SFTP settings that are specified on the FileInput and FileOutput nodes.

The settings that you specify by using an FtpServer configurable service are read and validated when the message flow starts, and are used to configure any SFTP connections that are made for the node. The configurable service can override any or all of the remote transfer properties on the **FTP** tab of the FileInput and FileOutput nodes. For more information about the settings that you can specify with an FtpServer configurable service, see FtpServer configurable service properties.

You can configure strict host key checking and specify your own known hosts file, or you can turn off strict host key checking and use the known hosts files that are created and managed by the broker.

Multiple configurable services can specify the same host and port, even with different known hosts files. FTP defaults to port 21 and SFTP defaults to port 22, which is the SSH default port. If you set the port and specify an FTP connection to an SFTP server (or specify an SFTP connection to an FTP server) a connection error occurs and a message is added to the event log.

You can use the FtpServer configurable service to configure the following SFTP settings:

- Cipher used for SSH/SFTP communication
- Compression level
- Strict known host checking
- Protocol (FTP/SFTP) for nodes to use for remote file transfer
- Location of a known hosts file when strict known host checking is set to Yes

1. Use the mqsicreateconfigurableservice command to create an FtpServer configurable service with the required parameter values. For more information about creating the FtpServer configurable service, see FtpServer configurable service properties.
2. In the FileInput and FileOutput nodes, specify the name of the FtpServer configurable service in the Remote server and port property on the **FTP** tab. Use the mqsichangeproperties and mqsireportproperties commands to change or view the properties of the configurable service.

Known host checking

Use known host checking to control which hosts the broker can connect to, and to verify the identity of those hosts.

Known host checking enables the broker to protect the messages in the message flow from unauthorized attempts to intercept the data (sometimes known as *man-in-the-middle attacks*). Known hosts files contain the SSH keys of the hosts to which the broker can connect. On z/OS systems, known hosts files and SSH identity files are stored in EBCDIC format, and on other operating systems they are stored in ASCII format.

Before it connects to a host to receive or transfer a file (using the FileInput or FileOutput nodes), the broker checks the host key against the keys that are stored in the known hosts file. If the host key does not match an existing entry for the host in the known hosts file, the connection fails. If the host is new (and has no entry in the known hosts file), the result depends on whether strict known host checking is turned on or off.

You can specify whether strict known host checking is turned on or off by setting the **strictHostKeyChecking** parameter of either the mqsicreateconfigurableservice command or the FtpServer configurable service. For more information about the settings that you can specify with an FtpServer configurable service, see FtpServer configurable service properties.

Strict known host checking

When strict known host checking is turned on (by setting the **strictHostKeyChecking** parameter to *Yes*), the broker connects only to known hosts with valid SSH host keys that are stored in the known hosts file. If you use strict known host checking, you must create your own known hosts file, in OpenSSH format, containing the SSH keys of your trusted hosts. You specify the location of your known hosts file using the **knownHostsFile** parameter of the mqsicreateconfigurableservice command. When the broker attempts to make a connection to a host, it checks the host key against the contents of the known hosts file. If the key is not in the known hosts file, the connection fails and a BIP3371 error occurs.

You can have multiple known hosts files and specify a different one for each node or configurable service. The known hosts files that you provide for strict known host checking are not modified by the broker.

Non-strict known host checking

When strict known host checking is turned off (by setting the **strictHostKeyChecking** parameter to *No*) the broker connects only to known hosts with valid keys or to new hosts to which it has not connected before. If the broker

attempts to connect to a new host (to which it has not connected previously), the broker makes the connection, accepts the host's SSH key, and stores it in the known hosts file. When the broker tries to connect to the same host on subsequent occasions, the host key is checked against the key stored in the known hosts file, and, if the key matches, the connection is made. However, if the host key is different from the one stored in the known hosts file, the connection attempt fails and a BIP3371 error occurs.

When strict known host checking is turned off, the known hosts file is managed by the broker. One broker-managed known hosts file exists for each execution group.

BIP3371 error message

The BIP3371 error message might indicate that there has been an unauthorized attempt to intercept a message. However, the connection failure (and resulting BIP3371 error message) might be caused by a change to the host's SSH key at some time following its first connection to the broker. For example, if the SSH server is reinstalled, it is assigned a new key, which is not found in the known hosts file and is therefore not accepted by the broker. As a result, the connection fails and the BIP3371 error message is shown.

If you know that the SSH host key has changed (as a result of a recent SSH server reinstallation, for example), you can solve the connection failure by modifying the known hosts file:

1. Stop the message flow.
2. Edit the known hosts file:
   - If you have strict known host checking turned on, correct the entry for the host in the known hosts file that you specified in the **knownHostsFile** parameter of the mqsicreateconfigurableservice command.
   - If you have strict known host checking turned off, remove the host's entry from the broker-managed known hosts file. The known hosts file is in the `\components\`*`BROKERNAME-NAME`*`\`*`EG-UID`*`\config\known_hosts` subdirectory of the directory in which WebSphere Message Broker is installed. For example, on Windows the default directory is `C:\Documents and Settings\All Users\Application Data\IBM\MQSI\components\`*`BROKERNAME-NAME`*`\`*`EG-UID`*`\config\known_hosts`. On UNIX the default directory is `/var/mqsi/components/`*`BROKERNAME-NAME`*`/`*`EG-UID`*`/config/known_hosts`.

     When the broker attempts to reconnect, it adds the new host key to the known hosts file.
3. Restart the message flow.

Part 4. Reference

Message flows

Use the reference information in this section to develop your message flows and related resources.

The following message flow reference information is available:
- "Message flow preferences"
- "Built-in nodes"
- "Description properties for a message flow" on page 1321
- "Configurable message flow properties" on page 1324
- "WebSphere Adapters properties" on page 1326
- "Validation properties" on page 1445
- "Parsing on demand" on page 1449
- "User-defined nodes" on page 1450
- "Supported code pages" on page 1450
- "WebSphere MQ connections" on page 1478
- "Listing database connections that the broker holds" on page 1478
- "Quiescing a database" on page 1478
- "Support for Unicode and DBCS data in databases" on page 1479
- "Data integrity in message flows" on page 1482
- "Exception list structure" on page 1482
- "Message flow porting" on page 1490
- "Monitoring message flows" on page 1490
- "Message flow accounting and statistics data" on page 1500
- "Coordinated message flows" on page 1516
- "Element definitions for message parsers" on page 1517
- "Message mappings" on page 1530
- "XML constructs" on page 1567
- "Data sources on z/OS" on page 1583

Message flow preferences

You can change preferences that determine properties of message flows when you create them.

Click **Window** → **Preferences**, then click **Message Flow** in the left pane.

Property	Type	Meaning
Default version tag	String	Provide the default version information that you want to be set in the message flow Version property when you create a message flow.

Built-in nodes

WebSphere Message Broker supplies built-in nodes that you can use to define your message flows.

The mode that your broker is working in can affect the types of node that you can use; see Restrictions that apply in each operation mode.

For information about each of these built-in nodes, use the following links. The nodes are listed in the categories under which they are grouped in the node palette, see "Message flow node palette" on page 8.

WebSphere MQ
- “MQInput node” on page 1083
- “MQOutput node” on page 1100
- “MQReply node” on page 1108
- “MQGet node” on page 1069
- “MQHeader node” on page 1079
- “MQOptimizedFlow node” on page 1099
- “MQeInput node” on page 1057
- “MQeOutput node” on page 1065

JMS
- “JMSInput node” on page 1020
- “JMSOutput node” on page 1033
- “JMSReply node” on page 1044
- “JMSHeader node” on page 1017
- “JMSMQTransform node” on page 1031
- “MQJMSTransform node” on page 1097

HTTP
- “HTTPInput node” on page 981
- “HTTPReply node” on page 988
- “HTTPRequest node” on page 991
- “HTTPHeader node” on page 977

Web Services
- “SOAPInput node” on page 1194
- “SOAPReply node” on page 1202
- “SOAPRequest node” on page 1204
- “SOAPAsyncRequest node” on page 1172
- “SOAPAsyncResponse node” on page 1181
- “SOAPEnvelope node” on page 1186
- “SOAPExtract node” on page 1189
- “RegistryLookup node” on page 1132
- “EndpointLookup node” on page 940

WebSphere Adapters
- “PeopleSoftInput node” on page 1115
- “PeopleSoftRequest node” on page 1119
- “SAPInput node” on page 1147
- “SAPRequest node” on page 1151
- “SiebelInput node” on page 1165
- “SiebelRequest node” on page 1168
- “TwineballInput node” on page 1297
- “TwineballRequest node” on page 1300

Routing
- “Filter node” on page 970
- “Label node” on page 1050
- “Publication node” on page 1126
- “RouteToLabel node” on page 1145
- “Route node” on page 1142
- “AggregateControl node” on page 878
- “AggregateReply node” on page 880
- “AggregateRequest node” on page 883
- “Collector node” on page 888

Transformation
- “Mapping node” on page 1052
- “XSLTransform node” on page 1311
- “Compute node” on page 894
- “JavaCompute node” on page 1013
- “PHPCompute node” on page 1122

Construction
- “Input node” on page 1012
- “Output node” on page 1112
- “Throw node” on page 1279
- “Trace node” on page 1289
- “TryCatch node” on page 1295
- “FlowOrder node” on page 975
- “Passthrough node” on page 1113
- “ResetContentDescriptor node” on page 1136

Database
- “Database node” on page 902
- “DataDelete node” on page 923
- “DataInsert node” on page 926
- “DataUpdate node” on page 930
- “Warehouse node” on page 1307
- “DatabaseRetrieve node” on page 907
- “DatabaseRoute node” on page 915
- “Extract node” on page 944

File
- “FileInput node” on page 946
- “FileOutput node” on page 959

Email
- “EmailOutput node” on page 933

TCPIP
- “TCPIPClientInput node” on page 1213
- “TCPIPClientOutput node” on page 1225
- “TCPIPClientReceive node” on page 1234
- “TCPIPServerInput node” on page 1245
- “TCPIPServerOutput node” on page 1256
- “TCPIPServerReceive node” on page 1264

IMS
- “IMSRequest node” on page 1004

Validation
- “Validate node” on page 1303
- “Check node” on page 885

Timer
- “TimeoutControl node” on page 1281
- “TimeoutNotification node” on page 1284

Additional protocols
- “SCADAInput node” on page 1155
- “SCADAOutput node” on page 1162

AggregateControl node

Use the AggregateControl node to mark the beginning of a fan-out of requests that are part of an aggregation.

This topic contains the following sections:

Purpose

Aggregation is an extension of the request/reply application model. It combines the generation and fan-out of a number of related requests with the fan-in of the corresponding replies, and compiles those replies into a single aggregated reply message.

The aggregation function is provided by the following three nodes:

- The AggregateControl node marks the beginning of a fan-out of requests that are part of an aggregation. It sends a control message that is used by the AggregateReply node to match the different requests that have been made. The information that is propagated from the Control terminal includes the broker identifier, the aggregate name property, and the timeout property. You must not change the aggregation information that is added to the message Environment by the AggregateControl node.
- The AggregateRequest node records the fact that the request messages have been sent. It also collects information that helps the AggregateReply node to construct the aggregated reply message. You must preserve the information that is added to the message Environment by the AggregateControl node, otherwise the aggregation fails.
- The AggregateReply node marks the end of an aggregation fan-in. It collects replies and combines them into a single aggregated reply message.

This node creates the `LocalEnvironment.ComIbmAggregateControlNode` folder. This folder and the fields in it are for internal use by WebSphere Message Broker and you should not rely on their existence or values when developing your message flows.

The AggregateControl node is contained in the **Routing** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Look at the following samples to see how to use this node:

- Aggregation
- Airline Reservations

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

Terminals and properties

When you have put an instance of the AggregateControl node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The AggregateControl node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Out	The output terminal to which the original message is routed when processing completes successfully.
Control	The output terminal to which a control message is routed. The control message is sent to a corresponding AggregateReply node. The Control terminal is deprecated in Version 6.0; to use connections from the Control terminal, see "Using control messages in aggregation flows" on page 678.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The AggregateControl node Description properties are described in the following table:

Property	M	C	Default	Description
Node name	No	No	The node type (AggregateControl)	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The AggregateControl node Basic properties are described in the following table:

Property	M	C	Default	Description	mqsiapplybaroverride command property
Aggregate Name	Yes	Yes		A name that is used to associate the fan-out message flow with the fan-in message flow. This value must be contextually unique in a broker. This name is also used to identify an Aggregation configurable service (if one exists) to be used by the node.	aggregateName

Property	M	C	Default	Description	mqsiapplybaroverride command property
Timeout (sec)	Yes	No	0	The amount of time, in seconds, that it waits for replies to arrive at the fan-in. The default value is zero; if you accept this default value, the timeout is disabled for fan-outs from this node (that is, it waits for replies indefinitely). If not all responses are received, the message flow continues to wait, and does not complete. Set a value greater than zero to ensure that the message flow can complete, even if not all responses are received. For further information about timeouts, see "AggregateReply node." z/OS On z/OS, if the Timeout property is not set to zero, set the queue manager parameter **EXPRYINT** to 5. The value specified by the Timeout (sec) property is overridden by the value set in the **timeoutSeconds** property of the Aggregation configurable service, if it is set. Timeout values specified by the node and the configurable service are overridden by a timeout value defined in the message, at the location specified by the Timeout location property of the AggregateControl node.	

The AggregateControl node Advanced properties are described in the following table:

Property	M	C	Default	Description
Timeout location	No	No	`'$LocalEnvironment/Aggregation/Timeout'`	The location in the message tree where the aggregation timeout value is defined. The value specified in the message tree overrides the Timeout (sec) property of the AggregateControl node and the **timeoutSeconds** property of the Aggregation configurable service.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

AggregateReply node

Use the AggregateReply node to mark the end of an aggregation fan-in. This node collects replies and combines them into a single compound message.

This topic contains the following sections:

Purpose

Aggregation is an extension of the request/reply application model. It combines the generation and fan-out of a number of related requests with the fan-in of the corresponding replies, and compiles those replies into a single aggregated reply message.

The aggregation function is provided by the following three nodes:

- The AggregateControl node marks the beginning of a fan-out of requests that are part of an aggregation. It sends a control message that is used by the AggregateReply node to match the different requests that have been made. The information that is propagated from the Control terminal includes the broker identifier, the aggregate name property, and the timeout property. The aggregation information that is added to the message Environment by the AggregateControl node must not be changed.
- The AggregateRequest node records the fact that the request messages have been sent. It also collects information that helps the AggregateReply node to construct the aggregated reply message. The information that is added to the message Environment by the AggregateRequest must be preserved, otherwise the aggregation fails.
- The AggregateReply node marks the end of an aggregation fan-in. It collects replies and combines them into a single aggregated reply message.

The AggregateReply node is contained in the **Routing** drawer of the palette, and is represented in the workbench by the following icon:

When incoming messages are stored by the AggregateReply node before all responses for the aggregation are received, the persistence of the message determines whether the message is retained across a restart.

If, during an aggregation, one or more of the response messages are not received by the AggregateReply node, the normal timeout or unknown message processing deals with the responses that have been received already.

The MQMD.Expiry value of each AggregateReply message is set to -1 in the compound output message. This value is set because the MQMD.Expiry value has no meaning once the reply message is no longer on the WebSphere MQ Transport and has been stored by the broker during the aggregation process.

Using this node in a message flow

Look at the following samples to see how to use this node:

- Aggregation
- Airline Reservations

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

Terminals and properties

When you have put an instance of the AggregateReply node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The AggregateReply node terminals are described in the following table.

Terminal	Description
Control	The input terminal that accepts control messages that are sent by a corresponding AggregateControl node. The Control terminal is deprecated in Version 6.0; to use connections to the Control terminal, see "Using control messages in aggregation flows" on page 678.
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if a failure is detected during processing.
Unknown	The output terminal to which messages are routed when they cannot be identified as valid reply messages.
Out	The output terminal to which the compound message is routed when processing completes successfully.
Timeout	The output terminal to which the incomplete compound message is routed when the timeout interval that is specified in the corresponding AggregateControl node has expired.
Catch	The output terminal to which the message is routed if an exception is thrown downstream and then caught by this node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the AggregateReply node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type (AggregateReply)	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The AggregateReply node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Aggregate Name	Yes	Yes		A name that is used to associate the fan-in message flow with the fan-out message flow. This value must be contextually unique within a broker.	aggregateName

Property	M	C	Default	Description	mqsiapplybaroverride command property
Unknown Message Timeout	No	No	0	The amount of time, in seconds, for which messages that cannot be identified as replies are held before they are propagated to the Unknown terminal. The default value is zero; if you accept this default value, the timeout is disabled, and unknown messages are propagated to the Unknown terminal upon receipt. z/OS On z/OS, if the Unknown Message Timeout property is not set to zero, set the queue manager parameter **EXPRYINT** to 5.	
Transaction Mode	Yes	No	Selected	This property defines the transactional characteristics of this message: • If you select the check box (the default), the subsequent message flow is under transaction control. This setting remains true for messages that derive from the output message and are produced by an MQOutput node, unless the MQOutput node explicitly overrides the transaction status. No other node can change the transactional characteristics of the output message. • If you clear the check box, the subsequent message flow is not under transaction control. This setting remains true for messages that derive from the output message and are produced by an MQOutput node, unless the MQOutput node has specified that the message must be put under sync point.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

AggregateRequest node

Use the AggregateRequest node to record the fact that request messages have been sent. This node also collects information that helps the AggregateReply node to construct the compound response message.

This topic contains the following sections:

Purpose

Aggregation is an extension of the request/reply application model. It combines the generation and fan-out of a number of related requests with the fan-in of the corresponding replies, and compiles those replies into a single aggregated reply message.

The aggregation function is provided by the following three nodes:

- The AggregateControl node marks the beginning of a fan-out of requests that are part of an aggregation. It sends a control message that is used by the AggregateReply node to match the different requests that have been made. The information that is propagated from the Control terminal includes the broker identifier, the aggregate name property, and the timeout property. The aggregation information that is added to the message Environment by the AggregateControl node must not be changed.
- The AggregateRequest node records the fact that the request messages have been sent. It also collects information that helps the AggregateReply node to construct the aggregated reply message. The information that is added to the message Environment by the AggregateRequest node must be preserved, otherwise the aggregation fails.
- The AggregateReply node marks the end of an aggregation fan-in. It collects replies and combines them into a single aggregated reply message.

The AggregateRequest node is contained in the **Routing** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Look at the following samples to see how to use this node:

- Aggregation
- Airline Reservations

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

Terminals and properties

When you have put an instance of the AggregateRequest node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The AggregateRequest node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts messages sent as part of an aggregate request.
Out	The output terminal to which the input message is routed when processing completes successfully.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The AggregateRequest node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type (AggregateRequest)	The name of the node
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The AggregateRequest node Basic property is described in the following table.

Property	M	C	Default	Description
Folder Name	Yes	No		The name that is used as a folder in the AggregateReply node's compound message to store the reply to this request. You must enter a value for this property, but the value does not need to be unique.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Check node

Use the Check node to compare the template of a message that is arriving on its input terminal with a message template that you supply when you configure the Check node.

Attention: The Check node is deprecated in WebSphere Message Broker Version 6.0 and subsequent versions. Although message flows that contain a Check node remain valid, redesign your message flows where possible to replace Check nodes with Validate nodes.

This topic contains the following sections:

Purpose

The message domain, message set, and message type of the message are collectively called the *message template*. The domain defines the parser that is used for the message. The set is the message set to which the message belongs. The type is the structure of the message itself. You can check the incoming message against one or more of these properties. The message property is checked only if you select its corresponding Check property, which means that a message property that contains a null string can be compared.

If the message properties match the specification, the message is propagated to the Match terminal of the node. If the message properties do not match the specification, the message is propagated to the Failure terminal. If the Failure terminal is not connected to some failure handling processing, an exception is generated.

The Check node is contained in the **Validation** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Use the Check node to ensure that the message is routed appropriately through the message flow. For example, you can configure the node to direct a message that requests stock purchases through a different route from that required for a message that requests stock sales.

Another example of this node's use is for the receipt of electronic messages from staff at your head office. These messages are used for multiple purposes; for example, to request technical support or stationery, or to advise you about new customer leads. These messages can be processed automatically because your staff complete a standard form. If you want these messages to be processed separately from other messages received, use the Check node to ensure that only staff messages with a specific message type are processed by this message flow.

Terminals and properties

When you have put an instance of the Check node into a message flow, node into a message flow, you can configure it. For more information, see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The Check node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if the incoming message does not match the specified properties.
Match	The output terminal to which the message is routed if the incoming message matches the specified properties.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Check node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	Check	The name of the node
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Check node Basic properties are described in the following table.

Property	M	C	Default	Description
Domain	No	No		The name of the domain.
Check domain	Yes	No	Cleared	This property checks that a message belongs to a particular domain. To check the parser that is to be used for the incoming message, select this check box and select one of the values from the Domain list.
Set	No	No		The message set to which the incoming message belongs. If you are using the MRM, IDOC, or XMLNSC parser, check that the incoming message belongs to a particular message set by selecting Check set and entering the name of the message set in Set. Leave Set clear for other parsers. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Check set	Yes	No	Cleared	If you select this check box, the incoming message is checked against the Set property.
Type	No	No		The message name. If you are using the MRM parser, check that the incoming message is a particular message type by selecting Check type and entering the name of the message in Type. Leave Type clear for other parsers.
Check type	Yes	No	Cleared	If you select this check box, the incoming message is checked against the Type property.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Collector node

Use the Collector node to create message collections based on rules that you configure in the node.

This topic contains the following sections:

Purpose

Use the Collector node to create a message collection from one or more sources based on configurable criteria. For example, you might need to extract, combine, and transform information from three different sources. The messages from these different sources might arrive at the input terminals at different times and in an unknown order. A collection is defined by configuring an event handler for each input terminal. Each event handler controls the acceptance of a message into a collection according to the following properties:

- Number of messages
- Collect messages for a set period
- Match the contents of a correlation path
- Match the contents against a correlation pattern

The correlation properties allow collections to be made according to the content of the messages. The content is specified by using an XPath expression. The Collector node ensures that each collection contains an identical correlation string across all its inputs. For more information about XPath 1.0 query syntax, see W3C XPath 1.0 Specification.

A message collection is created when the first message arrives at any of the dynamic input terminals on the Collector node. Message collections are stored on a WebSphere MQ queue.

When the conditions set by the event handlers for a message collection have been met, the message collection is complete and ready to be propagated. For example, if you set the event handlers on the Collector node to wait for two messages from each input terminal, the message collection is complete when two messages have been received by every terminal. When the next message arrives on an input terminal, it is added to a new message collection. You can select from a number of options to determine how the propagation of the message collection is coordinated. You can specify that the message collection is propagated automatically for processing, or alternatively that the message collection is propagated when a control message is received.

You can also set an expiry timeout for message collections that fail to be completed in a satisfactory time by using a property on the Collector node. The timeout starts when the first message is added to a message collection. If the timeout expires before the message collection is complete, the incomplete message collection is propagated to the **Expire** terminal. Set a value for the collection expiry to ensure that incomplete message collections do not remain stored on a queue indefinitely. Add appropriate processing to your message flow to handle incomplete message collections.

The Collector node is contained in the **Routing** drawer of the message flow node palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Look at the following sample to see how to use this node:

- Collector Node

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

Use the Collector node to group messages from different input sources for further processing. A message collection can be processed by the following nodes only:

- Compute
- JavaCompute

Errors might be generated if you try to process a message collection with a node that does not support message collections.

The Collector node has one static input terminal, **Control**, and four static output terminals: **Out**, **Expire**, **Failure** and **Catch**. These static terminals are always present on the node. In addition to the static input and output terminals, you can add dynamic input terminals for each input source that you want to use with the Collector node.

You can add and configure as many input terminals as required to the Collector node. You can configure the properties of each input terminal separately to control how the messages received on each input terminal are added to the appropriate message collection.

You can use the **Control** terminal to trigger the output of completed message collections from the Collector node. Configure the Event coordination property to set the behavior of the Collector node when messages are received on the **Control** terminal.

When a message collection is successfully completed, it is ready to be propagated to the **Out** terminal. If a value greater than zero is set on the Collection expiry property, any incomplete message collections are propagated to the **Expire** terminal.

A new transaction is created when a message collection is complete, and is propagated to the next node. Any exceptions that are caught from downstream nodes cause the message collection to be propagated to the **Catch** terminal on the

Collector node, together with the exception list. If the **Catch** terminal is not connected to any other nodes, the transaction is caused to roll back. Messages in the message collection are backed out onto the queue of the Collector node. The exception list is written to the system log. This step is repeated until the message collection is successfully processed. To avoid an exception that causes the message collection to fail to be propagated successfully, ensure that you connect the **Catch** terminal to a flow to handle any exceptions. Also, ensure that you set an expiry timeout to propagate incomplete message collections.

Note: Any exceptions that occur downstream of the Collector node are routed to the **Catch** terminal. The exception is not processed any further upstream because the completion of the message collection in the Collector node is the start of the transaction. This behavior is like the AggregateReply node. Do not connect a Throw node to the **Catch** terminal of the Collector node, because control is returned to the same **Catch** terminal.

If you use additional instances of a message flow or multiple inputs to the Collector node, you can use the Correlation path and Correlation pattern properties to ensure that related messages are added to the same message collection. If you use additional instances, or multiple inputs to the Collector node the order of messages in the message collection can be unpredictable. The order of messages is also unpredictable if you use WebSphere MQ cluster queues as inputs to the Collector node.

Configuring the Collector node

When you have put an instance of the Collector node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those properties that do not have a default value defined) are marked with an asterisk.

On the **Advanced** tab, use the Persistence Mode property to specify whether messages are stored persistently on the queues of the Collector node.

If you use additional instances of a message flow or multiple inputs to the Collector node, you can use the Correlation path and Correlation pattern properties to ensure that related messages are added to the same message collection. If you use additional instances, or multiple inputs to the Collector node the order of messages in the message collection can be unpredictable. The order of messages is also unpredictable if you use WebSphere MQ cluster queues as inputs to the Collector node.

Terminals and properties

The Collector node terminals are described in the following table.

Terminal	Description
Control	The static input terminal that accepts control messages. Any message received by the Control terminal is treated as a control message.
Out	The output terminal to which the complete message collection is routed if the received messages satisfy the configured conditions for the message collection.
Expire	The output terminal to which the incomplete message collection is routed if the received messages do not satisfy the configured conditions within the time specified on the Collection expiry property. If you have not set a value for the Collection expiry property this terminal is not used.

Terminal	Description
Failure	The output terminal to which the message collection is routed if a failure is detected during processing.
Catch	The output terminal to which the message collection is routed if an exception is thrown downstream and caught by this node.

The Collector node can have further dynamic input terminals. You can create numeric terminal labels for the Collector node; however, the Compute node does not support numeric labels. Therefore, when you are defining a custom terminal for the Collector node, ensure that the name begins with an alphabetic character.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Collector node Description properties are described in the following table:

Property	M	C	Default	Description
Node name	No	No	The node type, Collector	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Collector node has two types of Basic properties. You can set properties for each dynamic input terminal that you add to the Collector node in the Collection Definition table in the Basic properties. The properties in the Collection Definition table define the event handlers for the messages arriving on the individual input terminals. The properties that you can set for each of the dynamic input terminals are described in the following table:

Property	M	C	Default	Description
Terminal	Yes	No	The terminal name	Terminal is not a property of the node, but a label to show the name of the dynamic input terminal. Enter values for the event handler properties for each dynamic input terminal that you have added to the Collector node in the Collection Definition table.
Quantity	Yes	No	1	This property specifies the number of messages that the input terminal accepts to add to a message collection. The default value is 1; if you accept this default value, only one message is added to a collection. If a second message is received on the terminal, a new collection instance is created for it. If you select 0 (zero) or do not specify a value, there is no limit to the number of messages accepted. In this case, the value that is set on the Timeout property must be greater than zero.

Property	M	C	Default	Description
Timeout	Yes	No	0	This property specifies the maximum time in seconds for which the input terminal accepts messages. If you select 0 (zero), the timeout is disabled and there is no limit on the time to wait for messages. In this case, the value that is set on the Quantity property must be greater than zero.
Correlation path	No	No		Messages are only accepted into a message collection if they have the same correlation string. If the message has a different correlation string, it is offered to the next collection in the queue. If none of the collections accept the message, a new collection is created with correlation string set to the value of the correlation string in the message. Messages are grouped by the value from the correlation path. The correlation path is defined by using XPath. You can define your own correlation path by using XPath, or select from the following predefined paths: • $LocalEnvironment/Wildcard/WildcardMatch • $Root/MQMD/CorrelId • $LocalEnvironment/FileInput/Name • $Root/JMSTransport/Transport_Folders/Header_Values/ JMSCorrelationID If you define a value for Correlation path, you can optionally configure a Correlation pattern.
Correlation pattern	No	No		This property specifies a pattern to match the contents of a correlation path value against. You must set the Correlation path property before you set the value for the Correlation pattern property. If you set the correlation pattern, you must use one * character, optionally surrounded by other text. For example, *.dat. If the correlation pattern is blank, the entire text from the correlation path must be matched by the incoming message.

The remaining Basic properties for the Collector node are shown in the following table:

Property	M	C	Default	Description	mqsiapplybaroverride command property
Collection name	No	No		This property specifies the name of the message collection. • If you set this property to contain the wildcard *, the wildcard is replaced by the correlation string from the relevant event handler. • If you leave this property blank or use * and the correlation string is empty, then the collection name is set to an empty string.	
Collection expiry	No	Yes		The amount of time, in seconds, that the Collector node waits for messages to arrive. After this time an incomplete message collection is expired and propagated to the Expire output terminal. If you set this property to zero, the collection expiry is disabled and the Collector node waits for messages indefinitely. Set a value greater than zero to ensure that the message collection is processed, even if not all messages are received. A warning is issued if this property is not set.	collectionExpiry

The Collector node Advanced properties are described in the following table:

Property	M	C	Default	Description
Persistence mode	No	No	The node type, Collector	This property specifies whether messages are stored on the queues of the Collector node persistently.
Event coordination	Yes	No	Disabled	This property specifies how messages received on the Control terminal for event coordination processing are handled in the Collector node. • If you accept the default value (Disabled), messages to the Control terminal are ignored and collections are propagated when they are complete. • If you select All complete collections, complete message collections are held on a WebSphere MQ queue. When a message is received on the control terminal, all complete message collections on the WebSphere MQ queue are propagated to the Out terminal. • If you select First complete collection, complete message collections are held on a WebSphere MQ queue. When a message is received on the control terminal, the first complete message collection on the WebSphere MQ queue is propagated to the Out terminal. The collections are propagated in the same order that they become complete. If the WebSphere MQ queue is empty when the message is received on the Control terminal, the next complete message collection is immediately propagated to the Out terminal.
Configurable service	No	Yes	None set	This property specifies the name of the Collector configurable service to be used by the Collector node. The properties set by the Collector configurable service override the equivalent properties set on the Collector node. For more information about the properties than you can set with this configurable service, see Configurable services properties.

The Instances properties of the Collector node are described in the following table. For a full description of these properties, see "Configurable message flow properties" on page 1324.

Property	M	C	Default	Description
Additional instances pool	No	Yes	Use Pool Associated with Message Flow	The pool from which additional instances are obtained. • If you select Use Pool Associated with Message Flow, additional instances are obtained from the message flow pool. • If you select Use Pool Associated with Node, additional instances are allocated from the additional instances of the node based on the number specified in the Additional instances property.
Additional instances	No	Yes	0	The number of additional instances that the node can start if the Additional instances pool property is set to Use Pool Associated with Node.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

You cannot use monitoring properties to configure transaction events on the "Collector node" on page 888. Use a monitoring profile instead; see "Configuring monitoring event sources using a monitoring profile" on page 149.

Compute node

Use the Compute node to construct one or more new output messages.

This topic contains the following sections:

Purpose

The output messages that you create in the Compute node might be created by modifying the information that is provided in the input message, or by using only new information which can be taken from a database or from other sources. Elements of the input message (for example, headers, header fields, and body data), its associated environment, and its exception list can be used to create the new output message.

Specify how the new messages are created by coding ESQL in the message flow ESQL resource file. For more information, see "Specifying ESQL" on page 896.

Use the Compute node to:
- Build a new message using a set of assignment statements
- Copy messages between parsers
- Convert messages from one code set to another
- Transform messages from one format to another

The Compute node is contained in the **Transformation** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Look at the following samples to see how to use this node:
- Airline Reservations
- Aggregation
- JMS Nodes
- Large Messaging
- Message Routing
- Scribble
- Timeout Processing
- Video Rental

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

Consider a message flow in which you want to give each order that you receive a unique identifier for audit purposes. The Compute node does not modify its input message; it creates a new, modified copy of the message as an output message. You can use the Compute node to insert a unique identifier for your order into the output message, which can be used by subsequent nodes in the message flow.

Configuring the Compute node

When you have put an instance of the Compute node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view.

All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

Configure the Compute node by:
1. "Defining database interaction"
2. "Specifying ESQL" on page 896
3. "Setting the mode" on page 897
4. "Validating messages" on page 900

Defining database interaction:

To access a database from this node:

- On the **Basic** tab, specify in Data Source the name by which the appropriate database is known on the system on which this message flow is to run. The broker connects to this database with user ID and password information that you have specified on the mqsicreatebroker, mqsichangebroker, or mqsisetdbparms command.

  z/OS On z/OS systems, the broker uses the broker started task ID, or the user ID and password that were specified on the mqsisetdbparms command JCL, BIPSDBP in the customization data set <hlq>.SBIPPROC.
- Select the Transaction setting from the list. The values are:
  - Automatic (the default). The message flow, of which the Compute node is a part, is committed if it is successful. That is, the actions that you define in the ESQL module are performed on the message and it continues through the message flow. If the message flow fails, it is rolled back. If you choose Automatic, the ability to commit or roll back the action of the Compute node on the database depends on the success or failure of the entire message flow.

– Commit. To commit the action of the Compute node on the database, irrespective of the success or failure of the message flow as a whole, select Commit. The database update is committed even if the message flow itself fails.

The value that you choose is implemented for the one or more database tables that you have added; you cannot select a different value for each table.

- To treat database warning messages as errors and have the node propagate the output message to the Failure terminal, select Treat warnings as errors. The check box is cleared initially.

  When you select the check box, the node handles all positive return codes from the database as errors and generates exceptions in the same way as it does for the negative, or more serious, errors.

  If you do not select the check box, the node treats warnings as typical return codes, and does not raise any exceptions. The most significant warning raised is `not found`, which can be handled as a typical return code safely in most circumstances.
- To force the broker to generate an exception when a database error is detected, select Throw exception on database error. The check box is selected initially.

  If you clear the check box, you must include ESQL to check for any database error that might be returned after each database call that you make (you can use SQLCODE and SQLSTATE to do this). If an error occurs, you must handle the error in the message flow to ensure the integrity of the broker and the database; the error is ignored if you do not handle it through your own processing because you have chosen not to invoke the default error handling by the broker. For example, you can include the ESQL THROW statement to throw an exception in this node, or you can use the Throw node to generate your own exception at a later point in the message flow.

Specifying ESQL:

Code ESQL statements to customize the behavior of the Compute node. For example, you can customize the node to create a new output message or messages, using input message or database content (unchanged or modified), or new data. For example, you might want to modify a value in the input message by adding a value from a database, and storing the result in a field in the output message.

Code the ESQL statements that you want in an ESQL file that is associated with the message flow in which you have included this instance of the Compute node. The ESQL file, which by default has the name `<message_flow_name>.esql`, contains ESQL for every node in the message flow that requires it. Each portion of code that is related to a specific node is known as a module.

If an ESQL file does not already exist for this message flow, double-click the Compute node, or right-click the node and click **Open ESQL**. This action creates and opens a new ESQL file in the ESQL editor view. If you prefer, you can open the appropriate ESQL file in the Broker Development view and select this node in the Outline view.

If the file exists already, click **Browse** beside the ESQL Module property to display the Module Selection dialog box, which lists the available Compute node modules defined in the ESQL files that are accessible by this message flow (ESQL files can be defined in other, dependent, projects). Select the appropriate module and click **OK**. If no suitable modules are available, the list is empty.

If the module that you have specified does not exist, it is created for you and the editor displays it. If the file and the module exist already, the editor highlights the correct module.

If a module skeleton is created for this node in a new or existing ESQL file, it consists of the following ESQL. The default module name is shown in this example:

```
CREATE COMPUTE MODULE <flow_name>_Compute
       CREATE FUNCTION Main() RETURNS BOOLEAN
       BEGIN
              -- CALL CopyMessageHeaders();
              -- CALL CopyEntireMessage();
              RETURN TRUE;
       END;

       CREATE PROCEDURE CopyMessageHeaders() BEGIN
              DECLARE I INTEGER 1;
              DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
              WHILE I < J DO
                     SET OutputRoot.*[I] = InputRoot.*[I];
                     SET I = I + 1;
              END WHILE;
       END;

       CREATE PROCEDURE CopyEntireMessage() BEGIN
              SET OutputRoot = InputRoot;
       END;
END MODULE;
```

If you create your own ESQL module, you must create this skeleton exactly as shown except for the procedure calls and definitions (described later in this section). You can change the default name, but ensure that the name you specify matches the name of the corresponding node property ESQL Module. If you want the module name to include one or more spaces, enclose the name in double quotation marks in the ESQL Module property.

Add your own ESQL to customize this node after the BEGIN statement that follows CREATE FUNCTION, and before RETURN TRUE. You can use the two calls included in the skeleton, to procedures CopyEntireMessage and CopyMessageHeaders.

These procedures, defined following function Main, provide common functions that you might want when you manipulate messages. The calls in the skeleton are commented out; remove the comment markers if you want to use the procedure. If you do not want to use a procedure, remove both the call and the procedure definition from the module.

Setting the mode:

The Compute Mode property controls which components are used by default in the output message. Select the property to specify whether the Message, LocalEnvironment (previously specified as DestinationList), and Exception List components that are either generated in the node or contained in the incoming message are used.

This default value is used when the transformed message is routed to the Out terminal when processing in the node is completed. The default value is also used whenever a PROPAGATE statement does not specify the composition of its output message.

Those components that are not included in your selection are passed on unchanged; even if you modify those components, the updates are local to this node.

Conversely, those components that are included in the selection are not passed on and the updates that are made in the node persist.

The seven possible values that the Compute Mode property can take are listed in the following table.

Mode	Description
Message (the default)	The message is generated or passed through by the Compute node as modified in the node.
LocalEnvironment	The LocalEnvironment tree structure is generated or passed through by the Compute node as modified in the node.
LocalEnvironment And Message	The LocalEnvironment tree structure and message are generated or passed through by the Compute node as modified by the node.
Exception	The Exception List is generated or passed through by the Compute node as modified by the node.
Exception And Message	The Exception List and message are generated or passed through by the Compute node as modified by the node.
Exception and LocalEnvironment	The Exception List and LocalEnvironment tree structure are generated or passed through by the Compute node as modified by the node.
All	The message, Exception List, and LocalEnvironment are generated or passed through by the Compute node as modified by the node.

The value of the Compute Mode property specifies which new message trees are propagated from the Compute node. Therefore, for those message trees that are selected, the input messages are discarded unless they are explicitly copied into the new equivalent output message tree.

If `All` is selected, the Compute node is expecting to generate all three new message trees for the Root, LocalEnvironment, and ExceptionList by populating the OutputRoot, OutputLocalEnvironment, and OutputExceptionList. The input message trees are not passed to the output unless they are copied explicitly from the Input to the Output.

Therefore, if the Compute Mode property is set to `All`, you must code the following ESQL to allow the input trees to be propagated to the output terminal:

```
SET OutputRoot = InputRoot;
SET OutputLocalEnvironment = InputLocalEnvironment;
SET OutputExceptionList = InputExceptionList;
```

If the ESQL was `CopyEntireMessage()`, the LocalEnvironment and ExceptionList are not copied across and are not propagated to the output terminal; they are lost at that node in the message flow.

To produce a new or changed output message, and propagate the same LocalEnvironment and ExceptionList, set the Compute Mode property to Message

so that the LocalEnvironment and ExceptionList that are passed to the Compute or Mapping node, are passed on from the Compute node.

The Compute Mode applies only to propagation from the node. You can create all three output trees in a Compute or Mapping node and these can be manipulated and exist in the node. However, the Compute Mode determines whether such output trees are used on propagation from the node.

On propagation from the node, the following trees are propagated from the Compute or Mapping node for the following settings.

Compute Mode	**Trees propagated**
All	OutputRoot, OutputLocalEnvironment, OutputExceptionList
Message	OutputRoot, InputLocalEnvironment, InputExceptionList
LocalEnvironment	InputRoot, OutputLocalEnvironment, InputExceptionList
LocalEnvironment and Message	OutputRoot, OutputLocalEnvironment, InputExceptionList
Exception	InputRoot, InputLocalEnvironment, OutputExceptionList
Exception and Message	OutputRoot, InputLocalEnvironment, OutputExceptionList
Exception and LocalEnvironment	InputRoot, OutputLocalEnvironment, OutputExceptionList

Where an output tree is named, ESQL creates this message tree before propagation. If your ESQL does not create the tree, no tree is propagated for that correlation name, and the input tree is not used in its place because the Compute Mode property did not indicate this option. Therefore, dependent on Compute Mode property settings and your ESQL, you might delete a tree that was input to the node, because you did not transfer it to the output tree, or propagate a changed tree as you intended.

The converse is also true. If your ESQL interrogates the input trees and does not need to propagate these trees, the Compute Mode property value might mean that the message tree propagates when you do not intend it to. For example, you might not want to propagate the LocalEnvironment and ExceptionList from a Compute node, but because you selected `Message`, the input versions of the trees are propagated. Even if the ESQL explicitly deletes the OutputLocalEnvironment and OutputExceptionList, these changes are local to that node because the Compute Mode property setting causes the input trees to be propagated.

The Environment component of the message tree is not affected by the Compute Mode property setting. Its contents, if any, are passed on from this node in the output message.

Set this property to reflect the output message format that you require. If you select an option (or accept the default value) that does not include a particular part of the message, that part of the message is not included in any output message that is constructed.

The Compute node has both an input and output message, so that you can use ESQL to refer to fields in either message. You can also work with both

InputLocalEnvironment and OutputLocalEnvironment, and InputExceptionList and OutputExceptionList, as well as the input and output message bodies.

Validating messages:

Set the validation properties to define how the message that is produced by the Compute node is to be validated. These properties do not cause the input message to be validated. It is expected that, if such validation is required, the validation has already been performed by the input node or a preceding validation node.

For more details, see "Validating messages" on page 204 and "Validation properties" on page 1445.

Terminals and properties

The Compute node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the input message is routed if an unhandled exception occurs during the computation.
Out	The output terminal to which the transformed message is routed when processing in the node is completed. The transformed message might also be routed to this terminal by a PROPAGATE statement.
Out1	The first alternative output terminal to which the transformed message might be routed by a PROPAGATE statement.
Out2	The second alternative output terminal to which the transformed message might be routed by a PROPAGATE statement.
Out3	The third alternative output terminal to which the transformed message might be routed by a PROPAGATE statement.
Out4	The fourth alternative output terminal to which the transformed message might be routed by a PROPAGATE statement.

For the syntax of the PROPAGATE statement, see PROPAGATE statement.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file for deployment).

The Compute node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Compute node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Data Source	No	Yes		The ODBC data source name for the database that contains the tables to which you refer in the ESQL file that is associated with this message flow (identified in the ESQL Module property). You can specify only one data source for the node. If the ESQL that is associated with this node includes a PASSTHRU statement or SELECT function and a database reference, you must specify a value for the Data Source property.	dataSource
Transaction	Yes	No	Automatic	The transaction mode for the node. Valid options are `Automatic` and `Commit`. The property is valid only if you have selected a database table for input.	
ESQL Module	Yes	No	Compute	The name of the module in the ESQL file that contains the statements to run against the database and input and output messages.	
Compute Mode	Yes	No	`Message`	Choose from: • Message • LocalEnvironment • LocalEnvironment And Message • Exception • Exception And Message • Exception And LocalEnvironment • All For more information about setting the mode options, see "Setting the mode" on page 897.	
Treat warnings as errors	Yes	No	Cleared	If you select the check box, database SQL warnings are treated as errors.	
Throw exception on database error	Yes	No	Selected	If you select this check box, database errors cause the broker to throw an exception.	

The Parser Options properties for the Compute node are described in the following table.

Property	M	C	Default	Description
Use XMLNSC Compact Parser for XMLNS Domain	No	No	Cleared	Setting this property causes the outgoing MQRFH2 to specify the XMLNS instead of XMLNSC parser, allowing an external application to remain unchanged. If outgoing messages do not contain MQRFH2 headers this property has no effect.

The Validation properties of the Compute node are described in the following table.

For a full description of these properties, see "Validation properties" on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are None, Content and Value, Content, and Inherit.	validateMaster
Failure Action	No	No	Exception	This property controls what happens if a validation failure occurs. You can set this property only if Validate is set to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Database node

Use the Database node to interact with a database in the specified ODBC data source.

This topic contains the following sections:

Purpose

You define the nature of the interaction by coding ESQL statements that specify the data from the input message, and perhaps transform it in some way (for example, to perform a calculation), and assign the result to a database table.

You can set a property to control whether the update to the database is committed immediately, or deferred until the message flow completes, at which time the update is committed or rolled back, according to the overall completion status of the message flow.

You can use specialized forms of this node to:
- Update values within a database table (the DataUpdate node)
- Insert rows into a database table (the DataInsert node)
- Delete rows from a database table (the DataDelete node)
- Store the message, or parts of the message, in a warehouse (the Warehouse node)

The Database node is contained in the **Database** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Look at the following samples to see how to use this node:
- Airline Reservations
- Error Handler

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

Consider a situation in which you receive an order for 20 monitors. If you have sufficient numbers of monitors in your warehouse, you want to reduce the stock level on your stock database. You can use the Database node to check that you have enough monitors available, and reduce the value of the quantity field in your database.

Terminals and properties

When you have put an instance of the Database node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those properties that do not have a default value defined) are marked with an asterisk.

The terminals of the Database node are described in the following table. For more information about the PROPAGATE statement, including its syntax, see PROPAGATE statement

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the input message is propagated if a failure is detected during the computation. If you have selected Treat warnings as errors, the node propagates the message to this terminal even if the processing completes successfully.
Out	The output terminal to which the transformed message is routed when processing in the node is completed. The transformed message might also be routed to this terminal by a PROPAGATE statement.
Out1	The first alternative output terminal to which the transformed message might be routed by a PROPAGATE statement.
Out2	The second alternative output terminal to which the transformed message might be routed by a PROPAGATE statement.
Out3	The third alternative output terminal to which the transformed message might be routed by a PROPAGATE statement.
Out4	The fourth alternative output terminal to which the transformed message might be routed by a PROPAGATE statement.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the Database node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, Database	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Database node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Data Source	No	Yes		The ODBC data source name of the database that contains the tables to which you refer in the ESQL that is associated with this node (identified by the Statement property). This name identifies the appropriate database as it is known on the system on which this message flow is to run. The broker connects to this database with user ID and password information that you have specified on the mqsicreatebroker, mqsichangebroker, or mqsisetdbparms command. z/OS On z/OS systems, the broker uses the broker started task ID, or the user ID and password that are specified on the mqsisetdbparms command JCL, BIPSDBP in the customization data set <hlq>.SBIPPROC. If the ESQL that is associated with this node includes a PASSTHRU statement or SELECT function and a database reference, you must specify a value for the Data Source property.	dataSource

Property	M	C	Default	Description	mqsiapplybaroverride command property

Statement	Yes	No	Database	The name of the module in the ESQL file that contains the statements to use against the database. If you want the module name to include one or more spaces, enclose the name in double quotation marks.
				
				The ESQL file, which by default has the name `<message_flow_name>.esql`, contains ESQL for every node in the message flow that requires it. Each portion of code that is related to a specific node is known as a module. When you code ESQL statements that interact with tables, those tables are assumed to exist within this database. If they do not exist, a database error is generated by the broker at run time.
				
				Code ESQL statements to customize the behavior of the Database node in an ESQL file that is associated with the message flow in which you have included this instance of the Database node. If an ESQL file does not exist for this message flow, double-click the Database node, or right-click the node and click **Open ESQL** to create and open a new ESQL file in the ESQL editor view.
				
				If an ESQL file exists, click **Browse** beside the Statement property to display the Module Selection dialog box, which lists the available Database node modules that are defined in the ESQL files that are accessible by this message flow (ESQL files can be defined in other, dependent, projects). Select the appropriate module and click **OK**. If no suitable modules are available, the list is empty.
				
				If the module that you have specified does not exist, it is created for you and the editor displays it. If the file and the module exist, the editor highlights the correct module. If a module skeleton is created for this node in a new or existing ESQL file, it consists of the following ESQL. The default module name is shown in this example:
				`CREATE DATABASE MODULE <flow_name>_Database`
				`  CREATE FUNCTION Main() RETURNS BOOLEAN`
				`  BEGIN`
				`  RETURN TRUE;`
				`  END;`
				`END MODULE;`
				
				If you create your own ESQL module, create exactly this skeleton. You can update the default name, but ensure that the name that you specify matches the name of the corresponding node property Statement.
				
				To customize this node, add your own ESQL after the BEGIN statement and before RETURN TRUE. You can use all the ESQL statements including SET, WHILE, DECLARE, and IF in this module, but (unlike the Compute node) the Database node propagates, unchanged, the message that it receives at its input terminal to its output terminal. Therefore, like the Filter node, you have only one message to refer to in a Database node.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Transaction	Yes	No	Automatic	The transaction mode for the node. The values are: • Automatic (the default). The message flow, of which the Database node is a part, is committed if it is successful; that is, the actions that you define in the ESQL module are performed and the message continues through the message flow. If the message flow fails, it is rolled back. If you select Automatic, the ability to commit or roll back the action of the Database node on the database depends on the success or failure of the entire message flow. • Commit. To commit all uncommitted actions that are performed in this message flow on the database that is connected to this node, irrespective of the success or failure of the message flow as a whole, select Commit. The changes to the database are committed even if the message flow itself fails.	
Treat Warnings as Errors	Yes	No	Cleared	For database warning messages to be treated as errors, and for the node to propagate the output message to the Failure terminal, select Treat Warnings as Errors. The check box is cleared initially. When you select the check box, the node handles all positive return codes from the database as errors, and generates exceptions in the same way that it does for the negative, or more serious, errors. If you do not select the check box, the node treats warnings as normal return codes, and does not raise an exception. The most significant warning raised is `not found`, which can be handled safely as a typical return code in most circumstances.	
Throw Exception on Database Error	Yes	No	Selected	For the broker to generate an exception when a database error is detected, select Throw Exception on Database Error. The check box is selected initially. If you clear the check box, include ESQL to check for database errors that might be returned after each database call that you make (you can use SQLCODE and SQLSTATE to get this information). If an error has occurred, you must handle the error in the message flow to ensure the integrity of the broker and the database; the error is ignored if you do not handle it through your own processing because you have chosen not to use the default error handling by the broker. For example, you can include the ESQL THROW statement to throw an exception in this node, or you can use the Throw node to generate your own exception at a later point.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

DatabaseRetrieve node

Use the DatabaseRetrieve node to ensure that information in a message is up to date.

This topic contains the following sections:

Purpose

Use the DatabaseRetrieve node to modify a message using information from a database. For example, you can add information to a message using a key that is contained in a message; the key can be an account number.

The DatabaseRetrieve node is contained in the **Database** drawer of the message flow node palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Look at the following sample to see how to use this node:

- Simplified Database Routing

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

Input parameter values that are acquired from message elements in the incoming message are supported for insertion into prepared statements that are used by this node. These values are acquired from name, value, and name-value elements in the incoming parsed input message. Elements are acquired initially in the form of a com.ibm.broker.plugin.MbElement object, therefore the range of supported Java object types that values can take is governed by this object's interface. When values are in the form of Java primitive types or Objects they are converted to their equivalent JDBC data type, as shown in the following table.

Java	JDBC
Integer	INTEGER
Long	BIGINT
Double	DOUBLE
BigDecimal	NUMERIC
Boolean	BIT
byte[]	VARBINARY or LONGVARBINARY
BitSet	VARBINARY or LONGVARBINARY
String	VARCHAR or LONGVARCHAR
MbTime	java.sql.Time
MbTimestamp	java.sql.Timestamp
MbDate	java.sql.Date

Values are used from an element only if the element is of a known type, and its value state is valid, otherwise an exception is issued. Output column values that are acquired in the result set from SQL queries that are carried out by this node are converted first into matching Java types, then into internal message element value types, as shown in the following table.

JDBC	Java	ESQL Type
SMALLINT	Integer	INTEGER
INTEGER	Integer	INTEGER
BIGINT	Long	DECIMAL
DOUBLE	Double	FLOAT
REAL	Double	FLOAT
FLOAT	Double	FLOAT
NUMERIC	BigDecimal	DECIMAL
DECIMAL	BigDecimal	DECIMAL
BIT	Boolean	BOOLEAN
BOOLEAN	Boolean	BOOLEAN
BINARY	byte[]	BLOB
VARBINARY	byte[]	BLOB
LONGVARBINARY	byte[]	BLOB
CHAR	String	CHARACTER
VARCHAR	String	CHARACTER
LONGVARCHAR	String	CHARACTER
TINYINT	byte[1]	BLOB
TIME	java.util.Date	TIME
TIMESTAMP	java.util.Date	TIMESTAMP
DATE	java.util.Date	DATE

You can route a message to the same location, whether a query is successful against a specified database, by wiring both of the non-failure output terminals to the same output location.

If an error is found in the XPath expression of a pattern, it is reported during validation in the workbench. The reported error message might include the incorrect expression string and its associated unique dynamic or static terminal name, or the string might be marked as broker in the table.

The DatabaseRetrieve node looks up values from a database and stores them as elements in the outgoing message assembly trees. The type of information that is obtained from the database in the form of output column values, which is acquired and passed back in the result set from SQL queries, is converted first into a matching Java type, then into an internal message element value type when it is finally stored in a location in am outgoing message assembly tree. If a message element already exists in the outgoing message tree, the new value overwrites the old value. If the target element does not exist, it is created, and the value is stored.

The node needs query information that is used to form an SQL select query, which can access multiple tables in a database using multiple test conditions. Sometimes, not all the information that you want to retrieve in a result set is in a single database table. To get the column values that you want, you might need to retrieve them from two or more tables. This node supports the use of SELECT statements that facilitate getting columns from one or more tables in a single result set. The normal join syntax that is supported is also referred to as *inner join*.

Inner join information that is collected to form a query includes a list of table qualified column values to retrieve and a list of test conditions, which form the WHERE clause of the SELECT statement. Table qualified column values can form the left hand operand in a test condition. Choose a comparison operator to apply to this operand and, optionally, specify an operand on the right to complete the test condition. The operator could be a null comparison test, in which case an operand on the right is not needed. The value of the operand on the right can be a database type (such as Integer, Boolean, or Long), another table qualified column, or a value that is acquired from an element in the incoming message, as expressed through an XPath 1.0 general expression.

The application of the expression must result in a single element, double, Boolean, or string being returned, otherwise an exception occurs. If the query returns multiple rows, the first row is chosen and the rest are ignored, unless the Multiple rows option is selected. In this case, all rows are processed, and values in those rows are used to update the outgoing message assembly trees.

It can be useful to combine a DatabaseRetrieve node with other message flow nodes. For example, you can use an XSLTransform node to manipulate data before or after the DatabaseRetrieve node is invoked.

The DatabaseRetrieve node has one input terminal (In) and three output terminals (Out, KeyNotFound, and Failure). If the message is modified successfully, it is routed to the Out terminal. If the message is not modified successfully or a failure is detected during processing, the message is routed to the Failure terminal. If no rows are returned in the result set following execution of a specified SQL select query, the original message is routed to the KeyNotFound terminal.

Making the JDBC provider service available to the DatabaseRetrieve node

The DatabaseRetrieve node constructs its JDBC connections using connection details that are stored in the broker's registry as a configurable service. JDBCProvider configurable services are supplied for all supported databases.

Use the mqsichangeproperties command to modify the settings of the supplied service for your chosen database, or create a new service using the mqsicreateconfigurableservice command. See Setting up a JDBC provider for type 4 connections for further information and assistance on working with JDBCProvider services. You must set up a different JDBCProvider service for each database to which you want to connect.

When you have defined the service, set the Data source name property of this node to the name of the JDBCProvider service; the attributes of the service are used to establish connections for the DatabaseRetrieve node.

You must stop and restart the broker for your changes to take effect, unless you intend to create a new execution group on the broker to which you will deploy the message flow that contains this node.

Using the Data Source Explorer view to query data sources

Use the Data Source Explorer view to discover the names of tables in a target database, and the names of any columns in those tables. You must import database definitions for your databases into the workbench before you can view them in the Data Source Explorer view; see "Adding database definitions to the workbench" on page 578.

1. Switch to the Broker Application Development perspective.
2. In the Data Source Explorer view, expand **Connections**. The database connections are listed.
3. Expand a database connection to list the databases, then expand the appropriate database.
4. Expand **Schemas** to list the schemas, then expand the appropriate schema.
5. Expand **Tables** to list all the tables.
6. Click a table to show its properties in the Properties view.
7. In the Properties view, click the **Columns** tab to view the column names.

Configuring the DatabaseRetrieve node

When you have put an instance of the DatabaseRetrieve node into a message flow, you can configure it. For more information, see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view.

All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

Example

The following example adds new elements (family name and wage) to the incoming message structure. This example uses a database table called Employee.

EmployeeNumber	FamilyName	FirstName	Salary
00001	Smith	John	20000
00002	Jones	Harry	26000
00003	Doe	Jane	31000

To make a copy of the incoming message, select Copy message. When this property is selected, the node always creates an outgoing message assembly that is based on the incoming message assembly, and these trees in the new outgoing message assembly are modified and propagated to the node's Out terminal. This behavior enables modification of the outgoing message tree itself ($OutputRoot), in addition to the other logical trees in the message assembly: $OutputLocalEnvironment, $OutputDestinationList, $OutputExceptionList and $Environment. If the logical trees only in the message assembly are to be modified by this node, for performance reasons do not select Copy message. When this property is not selected, the node always works against the original incoming message assembly, and it expects that no updates are attempted against the message tree. If an XPath expression in the Data elements table tries to update this message tree through a reference to $OutputRoot, an MbReadOnlyMessageException occurs. The incoming message is:

```
<EmployeeRecord>
 <EmployeeNumber>00001</EmployeeNumber>
</EmployeeRecord>
```

Here is an example of the Query elements table.

Table name	Column name	Operator	Value Type	Value
Employee	FamilyName			
Employee	Salary			
Employee	EmployeeNumber	=	Element	$OutputRoot/XMLNSC/ EmpRecord/EmpNumber

Here is an example of the Data elements table.

Column name	Message element
Employee.FamilyName	$OutputRoot/XMLNSC/EmployeeRecord/FamilyName
Employee.Salary	$OutputRoot/XMLNSC/EmployeeRecord/Wage

The DatabaseRetrieve node connects to the Employee database table and extracts the value to compare from each incoming message. The XPath expression that is used to navigate to the message body is `$InputBody/EmployeeRecord/ EmployeeNumber`. The SQL query is:

```
SELECT Employee.FamilyName, Employee.Salary
FROM Employee
WHERE EmployeeNumber=?
ORDER BY Employee.FamilyName ASC, Employee.Salary ASC
```

where *?* is the value that is retrieved from the incoming message, which is located through the Value property in the third row of the Query elements table, which has a Value Type of Element.

- If the value at this location is *00001*, information for John Smith is retrieved. The first data element row says get the value of the FamilyName column that is returned from the query, and insert it into a new element named "FamilyName" under EmployeeRecord. The second data element row says get the value of the Salary column that is returned from the query, and insert it into a new element named "Wage" under EmployeeRecord. The resulting outgoing message is:

```
<EmployeeRecord>
 <EmployeeNumber>00001</EmployeeNumber>
 <Surname>Smith</Surname>
 <Wage>20000</Wage>
</EmployeeRecord>
```

- If the value at this location is *00002*, information for Harry Jones is retrieved. The resulting outgoing message is:

```
<EmployeeRecord>
 <EmployeeNumber>00002</EmployeeNumber>
 <Surname>Jones</Surname>
 <Wage>26000</Wage>
</EmployeeRecord>
```

If you select the Multiple rows property, and details of both of the employees are returned from a query in the form of two rows in the result set, the resulting outgoing message is:

```
<EmployeeRecord>
 <EmployeeNumber>00001</EmployeeNumber>
 <Surname>Smith</Surname>
 <Wage>20000</Wage>
 <EmployeeNumber>00002</EmployeeNumber>
 <Surname>Jones</Surname>
 <Wage>26000</Wage>
</EmployeeRecord>
```

Validating messages

Set the Validation properties to define how the message that is produced by the DatabaseRetrieve node is to be validated. These properties do not cause the input message to be validated. It is expected that, if such validation is required, the validation has already been performed by the input node or a preceding validation node.

For more details, see “Validating messages” on page 204 and “Validation properties” on page 1445.

Terminals and properties

The DatabaseRetrieve node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Out	The output terminal to which the outgoing message is routed when it has been modified successfully.
KeyNotFound	The output terminal to which the original message is routed, unchanged, when the result set is empty.
Failure	The output terminal to which the message is routed if a failure is detected during processing.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The DatabaseRetrieve node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	DatabaseRetrieve	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The DatabaseRetrieve node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Data source name	Yes	Yes	DB2	The alias that is used to locate JDBCProvider service definition that is stored in the broker registry. The alias is used to locate and build the JDBC connection URL that is used to connect to a DBMS. The connection URL is driver specific, but it includes the database name to which to connect. If connection to the database is by a login account and password, the node also uses this property as a lookup key, through which these values can be acquired from an expected matching broker registry DSN entry. If the DBMS is password protected, define the **-n** parameter on the mqsisetdbparms command for the JDBC unique security key before you deploy the message flow that contains this DatabaseRetrieve node.	dataSource
Copy message	No	Yes	Cleared	This property indicates if a copy of the original incoming message is required because the message tree is to be updated, possibly in addition to logical trees in the message assembly. By default, this check box is cleared. For performance reasons, select this property only if the input message will be augmented.	
Multiple rows	No	Yes	Cleared	This property indicates if all rows are processed when a query returns multiple rows. If you select Multiple rows, all rows are processed, and values in those rows are used to update the outgoing message assembly trees. If you do not select this property, the first row is chosen and the rest are ignored.	
Query elements	Yes	No		A table of query elements that are used to compose a single SQL select statement. The table consists of five columns and one or more rows. The columns are Table name, Column name, Operator, Value Type, and Value. These five properties describe a query element, indicating a table qualified column value to be retrieved from a database. In this case, the element forms part of the SELECT and ORDER BY clauses in the generated query. Otherwise, the query element acts as a test condition that forms a predicate in the WHERE clause in the generated query.	
Table name	Yes	No		The name of a database table that forms part of the SQL select statement, including the schema name; for example, *myschema.mytable*.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Column name	Yes	No		The name of the column in the database table to be retrieved in the results set, as qualified by the value of the Table name property. This SELECT clause can refer to this name as a column value to return from a query or to be referenced in a test condition in the WHERE clause.	
Operator	Yes	No		A comparison operator to apply to an operand on the left (the table column that is specified in the row's first two columns) and optionally a value to apply to an operand on the right. If you specify an Ascending 'ASC' or Descending 'DESC' operator value for this property, this row signifies the declaration of a table qualified column that forms part of the SELECT and ORDER BY clauses in the generated query and optionally can be referenced in future rows as a value to an operand on the right.	
Value Type	Yes	No		A value that is either set to None, or that indicates the type of value that is expressed in the last column of this row. If this property is not set to None, it refers to a row that describes a test condition in the WHERE clause of the SQL select statement.	
Value	Yes	No		A value that is either set to None, or that specifies one of a specified set of property types as expressed by the Value Type property. For example, if the Value Type property is set to Element, the Value property collects an XPath 1.0 general expression. The value that is returned from the expression when it is applied to the node's incoming message is used as value of the operand on the right to be compared through this predicate. The compared value of the operand on the right must match the type that is retrieved for the table column that is compared against as the operand on the left. Complex expressions are possible, where zero or more values can be acquired from the incoming message, and manipulated to formed a single value for comparison. For example, the sum of multiple field values in the incoming message can be calculated by a general expression that is presented for a value type of Element.	

The DatabaseRetrieve node Data elements table properties are described in the following table.

Property	M	C	Default	Description
Data elements	Yes	No		A list of data elements. A data element is described by the Column name and Message element properties.
Column name	Yes	No		The name of the database column from which to obtain the element value. The list of names is updated dynamically based on the column entries that are entered in the Query elements table.
Message element	Yes	No		An XPath 1.0 read-write path expression that describes the path location of a message element. The message element is where the database value is stored. The XPath expression must evaluate to a single element in the message.

The DatabaseRetrieve node Validation properties are described in the following table.

For a full description of these properties, see "Validation properties" on page 1445.

Property	M	C	Default	Description
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are None, Content and Value, Content, and Inherit.
Failure action	No	No	Exception	This property controls what happens if a validation failure occurs. You can set this property only if Validate is set to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

DatabaseRoute node

Use the DatabaseRoute node to route messages using information from a database in conjunction with XPath expressions.

This topic contains the following sections:

Purpose

The DatabaseRoute node uses a collection of named column values from a selected database row and synchronously applies one or more XPath expressions to these acquired values to make routing decisions.

For more information about XPath 1.0 query syntax, see W3C XPath 1.0 Specification.

The DatabaseRoute node is contained in the **Database** drawer of the message flow node palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Look at the following sample to see how to use this node:

- Simplified Database Routing

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

Input parameter values that are acquired from message elements in the incoming message are supported for insertion into prepared statements that are used by this node. These values are acquired from name, value, and name-value elements in the incoming parsed input message. Elements are acquired initially in the form of a com.ibm.broker.plugin.MbElement object, therefore the range of supported Java object types that values can take is governed by this object's interface. When values are in the form of Java primitive types or Objects they are converted into their equivalent JDBC data type, as shown in the following table.

Java	**JDBC**
Integer	INTEGER
Long	BIGINT
Double	DOUBLE
BigDecimal	NUMERIC
Boolean	BIT
byte[]	VARBINARY or LONGVARBINARY
BitSet	VARBINARY or LONGVARBINARY
String	VARCHAR or LONGVARCHAR
MbTime	java.sql.Time
MbTimestamp	java.sql.Timestamp
MbDate	java.sql.Date

Values are used from an element only if the element is of a known type, and its value state is valid, otherwise an exception is issued. Output column values that are acquired in the result set from SQL queries that are carried out by this node are converted first into matching Java types, then into internal message element value types, as shown in the following table.

JDBC	**Java**	**ESQL Type**
SMALLINT	Integer	INTEGER
INTEGER	Integer	INTEGER
BIGINT	Long	DECIMAL
DOUBLE	Double	FLOAT
REAL	Double	FLOAT
FLOAT	Double	FLOAT
NUMERIC	BigDecimal	DECIMAL
DECIMAL	BigDecimal	DECIMAL
BIT	Boolean	BOOLEAN

JDBC	Java	ESQL Type
BOOLEAN	Boolean	BOOLEAN
BINARY	byte[]	BLOB
VARBINARY	byte[]	BLOB
LONGVARBINARY	byte[]	BLOB
CHAR	String	CHARACTER
VARCHAR	String	CHARACTER
LONGVARCHAR	String	CHARACTER
TINYINT	byte[1]	BLOB
TIME	java.util.Date	TIME
TIMESTAMP	java.util.Date	TIMESTAMP
DATE	java.util.Date	DATE

You can route a message to the same location, whether a query is successful against a specified database, by wiring both of the non-failure output terminals to the same output location.

If an error is found in the XPath expression of a pattern, it is reported during validation in the workbench. The reported error message might include the incorrect expression string and its associated unique dynamic or static terminal name, or the string might be marked as broken within the table.

The node needs query information that is used to form an SQL select query, which can access multiple tables in a database using multiple test conditions. Sometimes, not all the information that you want to retrieve in a result set is in a single database table. To get the column values that you want, you might need to retrieve them from two or more tables. This node supports the use of SELECT statements that facilitate getting columns from one or more tables in a single result set. The typical join syntax that is supported is also referred to as *inner join*.

Inner join information that is collected to form a query includes a list of table qualified column values to retrieve and a list of test conditions, which form the WHERE clause of the SELECT statement. Table qualified column values can form the left operand in a test condition. Choose a comparison operator to apply to this operand and, optionally, specify a right operand to complete the test condition. The operator can be a null comparison test, in which no right operand is needed. The value of the right operand can be a database type (such as Integer, Boolean, or Long), another table qualified column, or a value that is acquired from an element in the incoming message, as expressed through an XPath 1.0 general expression.

When you deploy a DatabaseRoute node in a message flow, you can select a value that is associated with the Data Source Name property. The list of values contains references to existing IBM predefined JDBC provider entries that are defined when a broker is first created. These entries are incomplete, therefore you must modify them to access the data source definition with which you want to work. If an existing default IBM predefined JDBC provider is already referenced and in use by another JDBC database node that requires different settings, use the mqsicreateconfigurableservice command to specify a new JDBC provider entry.

Use the mqsideleteconfigurableservice command to delete any unwanted JDBC provider entries.

The DatabaseRoute node has one input terminal and a minimum of four output terminals: Match, keyNotFound, Default, and Failure. The keyNotFound, Default, and Failure output terminals are static, therefore they are always present on the node. The dynamic Match terminal is created automatically each time a new DatabaseRoute node is selected and used in the Message Flow editor. This behavior means that you do not always need to create this node's first dynamic output terminal, which is the minimum number of terminals needed for this node to operate. You can rename this dynamic terminal if "Match" is not an appropriate name.

A message is copied to the Default terminal if none of the filter expressions are true. If an exception occurs during filtering, the message is propagated to the Failure terminal. If the database query that is applied to the node's data source produces an empty result set (that is, no database rows are matched), a message is copied to the keyNotFound terminal. The DatabaseRoute node can define one or more dynamic output terminals. For all terminals, the associated filter expression is applied to the input message and, if the result is true, a copy of the message is routed to the specified terminal.

Each filter expression in the Filter table can be applied to:

- The input message
- The collection of named column values that are selected from a matched database row
- Both the input message and the returned column values
- Neither

because expressions can be any valid general XPath 1.0 expression.

As with the Route node, expressions are applied in the order that they are given in the filter table. If the result is true, a copy of the message is routed to its associated dynamic output terminal. If you set the Distribution Mode property to First, the application of all filter expressions might not occur.

The filter expression can fail if you compare a returned column value to a string literal. How a column entry is stored (for example, a fixed-length character field) determines what is returned for a specified column from the database. White space padding occurs for fixed-length character fields that are retrieved from a database, where the value that is stored is less than the specified column character storage length. In this case, padding occurs to the right of the character string that is returned, forming part of the string. You should remember this when comparing such a column value to a string literal, because an equality comparison expression might fail if the literal does not contain the exact same string, including padding characters.

For example, in a table called Employee, a database column called LastName that is defined as `char(10)` with the value 'Smith', is returned as 'Smith ', therefore the filter expression must be:

```
$Employee_LastName = 'Smith     '
```

which resolves to true. The expression:

```
$Employee_LastName = 'Smith'
```

resolves to false.

Alternatively, the XPath expression can use the following function:

```
Function: string normalize-space(string?)
```

The normalize-space function returns the argument string with white space normalized by stripping leading and trailing white space and replacing sequences of white space characters with a single space. Therefore the expression is:

```
normalize-space($Employee_LastName) = 'Smith'
```

Making the JDBC provider service available to the DatabaseRoute node

The DatabaseRoute node constructs its JDBC connections using connection details that are stored in the broker's registry as a configurable service. JDBCProvider configurable services are supplied for all supported databases. Use the mqsichangeproperties command to modify the settings of the supplied service for your chosen database, or create a new service using the mqsicreateconfigurableservice command. See Setting up a JDBC provider for type 4 connections for further information and assistance on working with JDBCProvider services. You must set up a different JDBCProvider service for each database to which you want to connect.

When you have defined the service, set the Data Source Name property of this node to the name of the JDBCProvider service; the attributes of the service are used to establish connections for the DatabaseRoute node.

You must stop and restart the broker for your changes to take effect, unless you intend to create a new execution group on the broker to which you will deploy the message flow that contains this node.

Using the Data Source Explorer view to query data sources

Use the Data Source Explorer view to discover the names of tables within a target database, and the names of any columns in those tables. You must import database definitions for your databases into the workbench before you can view them in the Data Source Explorer view; see “Adding database definitions to the workbench” on page 578

1. Switch to the Broker Application Development perspective.
2. In the Data Source Explorer view, expand **Connections**. The database connections are listed.
3. Expand a database connection to list the databases, then expand the appropriate database.
4. Expand **Schemas** to list the schemas, then expand the appropriate schema.
5. Expand **Tables** to list all the tables.
6. Click a table to show its properties in the Properties view.
7. In the Properties view, click the **Columns** tab to view the column names.

Configuring the DatabaseRoute node

When you have put an instance of the DatabaseRoute node into a message flow, you can configure it. For more information, see “Configuring a message flow node” on page 276. The properties of the node are displayed in the Properties view.

All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

Example

Consider the following example, which uses a database table called Employee.

EmployeeNumber	FamilyName	FirstName	Salary
00001	Smith	John	20000
00002	Jones	Harry	26000
00003	Doe	Jane	31000

The following DatabaseRoute node properties are set as specified:

Table Name	Column Name	Operator	Value Type	Value
Employee	FamilyName	ASC	None	None
Employee	Salary	ASC	None	None
Employee	EmployeeNumber	=	Element	$Body/ EmployeeRecord/ EmployeeNumber

The DatabaseRoute node connects to the Employee database table and extracts the value to compare from each incoming message. The XPath expression that is used to navigate to the message body is `$Body/EmployeeRecord/EmployeeNumber`. The SQL query is:

```
SELECT Employee.FamilyName, Employee.Salary
FROM Employee
WHERE EmployeeNumber=?
ORDER BY Employee.FamilyName ASC, Employee.Salary ASC
```

where *?* is the value that is retrieved from the incoming message. This value is located through the Value property in the third row of the query elements table, which has a value type of Element.

- If the value at this location is *00001*, information for John Smith is retrieved. The first XPath expression, which is associated with the out_expr1 dynamic terminal, is not met, therefore it does not meet its condition, and no message is propagated to the Out terminal. The second XPath expression is met, so that a copy of the input message is routed to the out_expr2 dynamic terminal.
- If the value at this location is *00002*, information for Harry Jones is retrieved. The first XPath expression, which is associated with the out_expr1 dynamic terminal, is met, so that a copy of the input message is routed to the out_expr1 terminal. The second XPath expression is not processed because the Distribution Mode property is set to First.

Terminals

The DatabaseRoute node terminals are described in the following table.

Terminal	Description
In	The static input terminal that accepts a message for processing by the node.
Match	A dynamic output terminal to which the original message can be routed when processing completes successfully. You can create additional dynamic terminals; see “Dynamic terminals” on page 921.
Default	The static output terminal to which the message is routed if no filter expression resolves to true.
keyNotFound	The static output terminal to which the message is copied if no database rows are matched.

Terminal	Description
Failure	The static output terminal to which the message is routed if a failure is detected during processing.

Dynamic terminals

The DatabaseRoute node can have further dynamic output terminals. Not all dynamic output terminals that are created on a DatabaseRoute node need to be mapped to an expression in the filter table. For unmapped dynamic output terminals, messages are never propagated to them. Several expressions can map to the same single dynamic output terminal. For more information about using dynamic terminals, see "Using dynamic terminals" on page 279.

Properties

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The DatabaseRoute node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, DatabaseRoute	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The DatabaseRoute node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Data Source Name	Yes	Yes	DB2	The alias that is used to locate JDBCProvider service definition that is stored in the broker registry. The alias is used to locate and build the JDBC connection URL for connection to a DBMS. The connection URL is driver specific, but it includes the database name to which to connect. If connection to the database is by a login account and password, the node also uses this property as a lookup key, through which these values can be acquired from an expected matching broker registry DSN entry. If the DBMS is password protected, define the **-n** parameter on the mqsisetdbparms command for the JDBC unique security key before you deploy the message flow that contains this DatabaseRoute node.	dataSource

Property	M	C	Default	Description	mqsiapplybaroverride command property
Query Elements	Yes	No		A table of query elements that are used to compose a single SQL select statement. The table consists of five columns and one or more rows. The columns are Table Name, Column Name, Operator, Value Type, and Value. These five properties describe a query element, indicating a table qualified column value to be retrieved from a database. In this case, the element forms part of the SELECT and ORDER BY clauses in the generated query. Otherwise, the query element acts as a test condition that forms a predicate within the WHERE clause in the generated query.	
Table Name	Yes	No		The name of a database table that forms part of the SQL select statement, including the schema name; for example, *myschema.mytable*.	
Column Name	Yes	No		The name of the column in the database table to be retrieved in the results set, as qualified by the value of the Table Name property. This SELECT clause can refer to this name as a column value to return from a query, or to be referenced in a test condition in the WHERE clause.	
Operator	Yes	No		A comparison operator to apply to a left operand (the table column that is specified in the row's first two columns) and optionally a right operand value. If you specify an Ascending 'ASC' or Descending 'DESC' operator value for this property, this row signifies the declaration of a table qualified column that forms part of the SELECT and ORDER BY clauses in the generated query and optionally can be referenced in future rows as a right operand value.	
Value Type	Yes	No		A value that is either set to None, or that indicates the type of value that is expressed in the last column of this row. If this property is not set to None, it refers to a row that describes a test condition in the WHERE clause of the SQL select statement.	
Value	Yes	No		A value that is either set to None, or that specifies one of a specified set of property types as expressed by the Value Type property. For example, if the Value Type property is set to Element, the Value property collects an XPath 1.0 general expression. The value that is returned from the expression when it is applied to the node's incoming message is used as the right operand value to be compared through this predicate. The compared value of the right operand must match the type that is retrieved for the table column that is compared against as the left operand. Complex expressions are possible, where zero or more values can be acquired from the incoming message, and manipulated to form a single value for comparison. For example, the sum of multiple field values in the incoming message can be calculated by a general expression that is presented for a value type of Element.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Distribution mode	No	Yes	All	This property specifies the routing behavior of this node when an inbound message matches multiple expressions. If the Distribution Mode property is set to First, the message is propagated to the first matching output terminal. If the Distribution Mode property is set to All, the message is propagated to all matching output terminals. If there is no matching output terminal, the message is propagated to the Default terminal.	

The DatabaseRoute node Filter Expression Table properties are described in the following table.

Property	M	C	Default	Description
Filter table	Yes	No		A table of filters (XPath 1.0 general expressions) and associated terminal names that define any extra filtering that is performed by this node. The table consists of two columns and one or more rows. You must have at least one row in this table so that the node can perform routing logic. As with the Route node, expressions are evaluated in the order in which they are displayed in the table. To improve performance, put the XPath expressions that are satisfied most frequently at the top of the filter table.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

DataDelete node

Use the DataDelete node to interact with a database in the specified ODBC data source.

This topic contains the following sections:

Purpose

The DataDelete node is a specialized form of the Database node, and the interaction is restricted to deleting one or more rows from a table within the database. You specify what is deleted by defining mapping statements that use the data from the input message to identify the action required.

You can set a property to control whether the update to the database is committed immediately, or deferred until the message flow completes, at which time the update is committed or rolled back, according to the overall completion status of the message flow.

The DataDelete node is contained in the **Database** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Consider a situation in which you are running a limited promotion. The goods are available only for the period of the promotion, and each customer can order only one item. When stocks of the sale goods run out, you want to remove their details from the stock database. When a message containing an order for the last item comes in, the DataDelete node is triggered to remove all the details about that item from the database.

Terminals and properties

When you have put an instance of the DataDelete node into a message flow, you can configure it. For more information, see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. (If you double-click the DataDelete node, you open the New Message Map dialog box.) All mandatory properties for which you must enter a value (those properties that do not have a default value defined) are marked with an asterisk.

The terminals of the DataDelete node are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the input message is propagated if a failure is detected during the computation. If you have selected Treat warnings as errors, the node propagates the message to this terminal even if the processing completes successfully.
Out	The output terminal that outputs the message following the execution of the database statement.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The DataDelete node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	DataDelete	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The DataDelete node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Data source	No	Yes		The ODBC data source name of the database that contains the tables to which you refer in the mappings that are associated with this node (identified by the Statement property). This name identifies the appropriate database as it is known on the system on which this message flow is to run. The broker connects to this database with user ID and password information that you have specified on the mqsicreatebroker, mqsichangebroker, or mqsisetdbparms command. z/OS On z/OS systems, the broker uses the broker started task ID, or the user ID and password that are specified on the mqsisetdbparms command JCL, BIPSDBP in the customization data set <hlq>.SBIPPROC.	dataSource
Statement	Yes	No	DataDelete	The name of the mapping routine that contains the statements that are to be executed against the database or the message tree. The routine is unique to this type of node. By default, the name that is assigned to the mapping routine is identical to the name of the mappings file in which the routine is defined. The default name for the file is the name of the message flow concatenated with the name of the node when you include it in the message flow (for example, `MFlow1_DataDelete.msgmap` for the first DataDelete node in message flow MFlow1). You cannot specify a value that includes spaces. If you click **Browse** next to this entry field, a dialog box is displayed that lists all the available mapping routines that can be accessed by this node. Select the routine that you want and click **OK**; the routine name is set in Statement. To work with the mapping routine that is associated with this node, double-click the node, or right-click the node and click **Open Mappings**. If the mapping routine does not exist, it is created for you with the default name in the default file. If the file exists, you can also open file *`flow_name_node_name`*`.msgmap` in the Broker Development view. A mapping routine is specific to the type of node with which it is associated; you cannot use a mapping routine that you have developed for a DataDelete node with a different node that uses mappings (for example, a DataInsert node). If you create a mapping routine, you cannot call it from another mapping routine, although you can call it from an ESQL routine. For more information about working with mapping files, and defining their content, see “Developing message mappings” on page 546.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Transaction	Yes	No	Automatic	The transaction mode for the node. The values are: • Automatic (the default). The message flow, of which the DataDelete node is a part, is committed if it is successful; that is, the actions that you define in the mappings are performed and the message continues through the message flow. If the message flow fails, it is rolled back. Therefore, if you select Automatic, the ability to commit or roll back the action of the DataDelete node on the database depends on the success or failure of the entire message flow. • Commit. To commit any uncommitted actions performed in this message flow on the database connected to this node, irrespective of the success or failure of the message flow as a whole, select Commit. The changes to the database are committed even if the message flow itself fails.	
Treat warnings as errors	Yes	No	Cleared	For database warning messages to be treated as errors, and the node to propagate the output message to the failure terminal, select Treat warnings as errors. The check box is cleared by default. When you select the check box, the node handles all positive return codes from the database as errors, and generates exceptions in the same way as it does for the negative, or more serious, errors. If you do not select the check box, the node treats warnings as typical return codes, and does not raise an exception. The most significant warning raised is `not found`, which can be handled as a typical return code safely in most circumstances.	
Throw exception on database error	Yes	No	Selected	For the broker to generate an exception when a database error is detected, select Throw exception on database error. The check box is selected by default. If you clear the check box, you must handle the error in the message flow to ensure the integrity of the broker and the database: the error is ignored if you do not handle it through your own processing because you have chosen not to run the default error handling by the broker. For example, you can connect the Failure terminal to an error processing subroutine.	

DataInsert node

Use the DataInsert node to interact with a database in the specified ODBC data source.

This topic contains the following sections:

Purpose

The DataInsert node is a specialized form of the Database node, and the interaction is restricted to inserting one or more rows into a table within the database. You specify what is inserted by defining mapping statements that use the data from the input message to define the action required.

You can set a property to control whether the update to the database is committed immediately, or deferred until the message flow completes, at which time the update is committed, or rolled back according to the overall completion status of the message flow.

The DataInsert node is contained in the **Database** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Consider a situation in which your company has developed a new product. The details about the product have been sent from your engineering department. Your message flow must extract details from the message and add them as a new row in your stock database.

Terminals and properties

When you have put an instance of the DataInsert node into a message flow, you can configure it. For more information, see “Configuring a message flow node” on page 276. The properties of the node are displayed in the Properties view. (If you double-click the DataInsert node, you open the New Message Map dialog box.) All mandatory properties for which you must enter a value (those properties that do not have a default value defined) are marked with an asterisk.

The terminals of the DataInsert node are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the input message is propagated if a failure is detected during the computation. If you have selected Treat warnings as errors, the node propagates the message to this terminal even if the processing completes successfully.
Out	The output terminal that outputs the message following the execution of the database statement.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The DataInsert node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	DataInsert	The name of the node.

Property	M	C	Default	Description
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The DataInsert node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Data source	No	Yes		The ODBC data source name of the database that contains the tables to which you refer in the mappings that are associated with this node (identified by the Statement property). This name identifies the appropriate database as it is known on the system on which this message flow is to run. The broker connects to this database with user ID and password information that you have specified on the mqsicreatebroker, mqsichangebroker, or mqsisetdbparms command. z/OS On z/OS systems, the broker uses the broker started task ID, or the user ID and password that are specified on the mqsisetdbparms command JCL, BIPSDBP in the customization data set <hlq>.SBIPPROC.	dataSource

Property	M	C	Default	Description	mqsiapplybaroverride command property
Statement	Yes	No	DataInsert	The name of the mapping routine that contains the statements that are to be executed against the database or the message tree. The routine is unique to this type of node. By default, the name that is assigned to the mapping routine is identical to the name of the mappings file in which the routine is defined. The default name for the file is the name of the message flow concatenated with the name of the node when you include it in the message flow (for example, `MFlow1_DataInsert.msgmap` for the first DataInsert node in message flow MFlow1). You cannot specify a value that includes spaces. If you click **Browse** next to this entry field, a dialog box is displayed that lists all the available mapping routines that can be accessed by this node. Select the routine that you want and click **OK**; the routine name is set in Statement. To work with the mapping routine that is associated with this node, double-click the node, or right-click the node and select **Open Mappings**. If the mapping routine does not exist, it is created for you with the default name in the default file. If the file exists, you can also open file *`flow_name_node_name`*`.msgmap` in the Broker Development view. A mapping routine is specific to the type of node with which it is associated; you cannot use a mapping routine that you have developed for a DataInsert node with a different node that uses mappings (for example, a DataDelete node). If you create a mapping routine, you cannot call it from another mapping routine, although you can call it from an ESQL routine. For more information about working with mapping files, and defining their content, see "Developing message mappings" on page 546.	
Transaction	Yes	No	Automatic	The transaction mode for the node. The values are: • Automatic (the default). The message flow, of which the DataInsert node is a part, is committed if it is successful. That is, the actions that you define in the mappings are taken and the message continues through the message flow. If the message flow fails, it is rolled back. Therefore if you select Automatic, the ability to commit or roll back the action of the DataInsert node on the database depends on the success or failure of the entire message flow. • Commit. To commit all uncommitted actions that are taken in this message flow on the database that is connected to this node, irrespective of the success or failure of the message flow as a whole, select Commit. The changes to the database are committed even if the message flow itself fails.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Treat warnings as errors	Yes	No	Cleared	For database warning messages to be treated as errors, and the node to propagate the output message to the failure terminal, select Treat warnings as errors. The check box is cleared by default. When you select the check box, the node handles all positive return codes from the database as errors and generates exceptions in the same way as it does for the negative, or more serious, errors. If you do not select the check box, the node treats warnings as typical return codes, and does not raise an exception. The most significant warning raised is `not found`, which can be handled as a typical return code safely in most circumstances.	
Throw exception on database error	Yes	No	Selected	For the broker to generate an exception when a database error is detected, select Throw exception on database error. The check box is selected by default. If you clear the check box, you must handle the error in the message flow to ensure the integrity of the broker and the database; the error is ignored if you do not handle it through your own processing because you have chosen not to run the default error handling by the broker. For example, you can connect the failure terminal to an error processing subroutine.	

DataUpdate node

Use the DataUpdate node to interact with a database in the specified ODBC data source.

This topic contains the following sections:

Purpose

The DataUpdate node is a specialized form of the Database node, and the interaction is restricted to updating one or more rows in a table within the database. You define what is updated by defining mapping statements that use the data from the input message to identify the action required.

You can set a property to control whether the update to the database is committed immediately, or deferred until the message flow completes, at which time the update is committed or rolled back according to the overall completion status of the message flow.

The DataUpdate node is contained in the **Database** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Consider a scenario in which you have added the details of a new product, a keyboard, to your stock database. Now you have received a message from the Goods In department that indicates that 500 keyboards have been delivered to your premises. You can use the DataUpdate node to change the quantity of keyboards in your database from zero to 500.

Terminals and properties

When you have put an instance of the DataUpdate node into a message flow, you can configure it. For more information, see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. (If you double-click the DataUpdate node, you open the New Message Map dialog box.) All mandatory properties for which you must enter a value (those properties that do not have a default value defined) are marked with an asterisk.

The terminals of the DataUpdate node are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the input message is propagated if a failure is detected during the computation. If you have selected Treat warnings as errors, the node propagates the message to this terminal even if the processing completes successfully.
Out	The output terminal that outputs the message following the execution of the database statement.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The DataUpdate node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	DataUpdate	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The DataUpdate node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Data source	No	Yes		The ODBC data source name of the database that contains the tables to which you refer in the mappings that are associated with this node (identified by the Statement property). This name identifies the appropriate database as it is known on the system on which this message flow is to run. The broker connects to this database with user ID and password information that you have specified on the mqsicreatebroker, mqsichangebroker, or mqsisetdbparms command. z/OS On z/OS systems, the broker uses the broker started task ID, or the user ID and password that are specified on the mqsisetdbparms command JCL, BIPSDBP in the customization data set <hlq>.SBIPPROC.	dataSource
Statement	Yes	No	DataUpdate	The name of the mapping routine that contains the statements that are to be executed against the database or the message tree. The routine is unique to this type of node. By default, the name that is assigned to the mapping routine is identical to the name of the mappings file in which the routine is defined. The default name for the file is the name of the message flow concatenated with the name of the node when you include it in the message flow (for example, `MFlow1_DataUpdate.msgmap` for the first DataUpdate node in message flow MFlow1). You cannot specify a value that includes spaces. If you click **Browse** next to this entry field, a dialog box is displayed that lists all available mapping routines that can be accessed by this node. Select the routine that you want and click **OK**; the routine name is set in Statement. To work with the mapping routine that is associated with this node, double-click the node, or right-click the node and click **Open Mappings**. If the mapping routine does not exist, it is created for you with the default name in the default file. If the file exists, you can also open file *flow_name_node_name*`.msgmap` in the Broker Development view. A mapping routine is specific to the type of node with which it is associated; you cannot use a mapping routine that you have developed for a DataUpdate node with a different node that uses mappings (for example, a DataInsert node). If you create a mapping routine, you cannot call it from another mapping routine, although you can call it from an ESQL routine. For more information about working with mapping files, and defining their content, see “Developing message mappings” on page 546.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Transaction	Yes	No	Automatic	The transaction mode for the node. The values are: • Automatic (the default). The message flow, of which the DataUpdate node is a part, is committed if it is successful. That is, the actions that you define in the mappings are performed and the message continues through the message flow. If the message flow fails, it is rolled back. Therefore, if you choose Automatic, the ability to commit or roll back the action of the DataUpdate node on the database depends on the success or failure of the entire message flow. • Commit. To commit all uncommitted actions that are performed in this message flow on the database that is connected to this node, irrespective of the success or failure of the message flow as a whole, select Commit. The changes to the database are committed even if the message flow itself fails.	
Treat warnings as errors	Yes	No	Cleared	For database warning messages to be treated as errors, and the node to propagate the output message to the Failure terminal, select Treat warnings as errors. The check box is cleared by default. When you select the check box, the node handles all positive return codes from the database as errors, and generates exceptions in the same way as it does for the negative, or more serious, errors. If you do not select the check box, the node treats warnings as normal return codes, and does not raise an exception. The most significant warning raised is `not found`, which can be handled safely as a normal return code in most circumstances.	
Throw exception on database error	Yes	No	Selected	For the broker to generate an exception when a database error is detected, select Throw exception on database error. The check box is selected by default. If you clear the check box, you must handle the error in the message flow to ensure the integrity of the broker and the database: the error is ignored if you do not handle it through your own processing, because you have chosen not to run the default error handling by the broker. For example, you can connect the Failure terminal to an error processing subroutine.	

EmailOutput node

Use the EmailOutput node to send e-mail messages to one or more recipients.

This topic contains the following sections:

Purpose

The EmailOutput node delivers an e-mail message from a message flow to an SMTP server that you specify.

You can configure the EmailOutput node using the node properties in the Message Broker Toolkit, or dynamically from the local environment and e-mail output header (EmailOutputHeader) that are associated with the message (for more information, see “Producing dynamic e-mail messages” on page 515).

The EmailOutput node is contained in the **Email** drawer of the message flow node palette, and is represented in the workbench by the following icon:

Look at the following sample to see how to use this node:

- E-mail

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

Configuring the EmailOutput node

When you have put an instance of the EmailOutput node into a message flow, you can configure it. For more information, see “Configuring a message flow node” on page 276. The properties of the node are displayed in the Properties view.

All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The following is a description of the four levels of configuration of the EmailOutput node:

- Option 1: Configure the EmailOutput node by using the node properties in the Message Broker Toolkit to send an e-mail with a statically-defined subject and text to a statically-defined list of recipients. The same e-mail is sent to the same recipients and it has no attachments. This method is useful when you want to test the EmailOutput node, or when notification alone is sufficient. For more details, see “Sending an e-mail” on page 513.
- Option 2: This option is the same as Option 1 but with the inclusion of an attachment. This option causes the e-mail message to be constructed as a MIME message. The subject, text, and list of recipients remains static, but the content of the attachment is sought dynamically from the message that is passed to the EmailOutput node at run time. The location of the attachment in the message is defined statically. For more details, see “Sending an e-mail with an attachment” on page 514.
- Option 3: This option allows for those properties in Options 1 and 2 to be optional, and to be overridden at run time by values that are specified in the local environment, the e-mail output header (EmailOutputHeader), or the body of the message. This option allows a dynamic e-mail message to be produced where the SMTP server, list of recipients, subject, text, and multiple attachments are all determined at run time. This option requires previous nodes in the message flow to construct these overrides. Where a text value is not specified in the node properties for the main body of the e-mail, the body of the message that is passed to the EmailOutput node is used. However, content that you set in the Email message text property overrides dynamically-generated text in the message body. For more details, see “Producing dynamic e-mail messages” on page 515.

- Option 4: This option passes a MIME message to the EmailOutput node. The EmailOutput node uses the MIME parser to write the MIME message to a bit stream. This message is then sent to the list of recipients in the SMTP header. Local environment overrides are not taken into consideration when a MIME message is passed. For more details, see "Sending a MIME message" on page 517.

E-mail output header

The e-mail output header (EmailOutputHeader) is a child of root. Values that you specify in this header override equivalent properties that you set on the node. Use the SMTP output header to specify any of the e-mail attributes, such as its recipients.

Location	Description
Root.EmailOutputHeader.To	A comma separated list of e-mail addresses.
Root.EmailOutputHeader.Cc	A comma separated list of e-mail addresses.
Root.EmailOutputHeader.Bcc	A comma separated list of e-mail addresses.
Root.EmailOutputHeader.From	A comma separated list of e-mail addresses.
Root.EmailOutputHeader.Reply-To	A comma separated list of e-mail addresses.
Root.EmailOutputHeader.Subject	The subject of the e-mail.

Local environment

Use the local environment to specify overrides to the SMTP server connection information and attachments.

Local environment	Description
Destination.Email.SMTPServer	The Server:Port of the SMTP server. Port is optional; if you do not specify it, the default value is 25.
Destination.Email.SecurityIdentity	The security identity for authentication with the SMTP server, which can be the name of the userid and password pair that is defined using the mqsisetdbparms command, or it can reference an external resource that has a securityIdentity attribute that references a userid and password that are defined using the mqsisetdbparms command. In both cases, the value is appended after the string "smtp::". For example, if you use the mqsisetdbparms command to create a userid and password of *smtp::myUseridPassword*, the securityIdentity that is specified on the node, or indirectly in an external resource, is *myUseridPassword*.

Local environment	Description

Destination.Email.BodyContentType	Identifies that the body of the e-mail message contains HTML rather than plain text. You can set this property to text/plain, text/html, or text/xml; text/plain is the default value.
	
	To set the content type for the body of the message, use the following notation.
	`SET OutputLocalEnvironment.Destination.Email.BodyContentType = "text/html"`
	
	To additionally set the character set (charset) in which the message body is sent, use the following notation.
	`SET OutputLocalEnvironment.Destination.Email.BodyContentType = "text/html; charset=utf-8"`
	
	This example sends a text/HTML e-mail with a charset of UTF-8.

Destination.Email.MultiPartContentType	The type of multipart, including related, mixed, and alternative. You can set any value here.
Destination.Email.Attachment.Content	Either the attachment (BLOB/text), or an XPath or ESQL expression that references an element; for example, an element in the message tree or local environment. The value of the referenced element is taken as the content of the attachment. • If the element is a BLOB, it is an attachment. • If the element is text, check to see if it can be resolved to another element in the message tree or local environment. If it can be resolved, use that element. If it cannot be resolved, add this element as the attachment.
Destination.Email.Attachment.ContentType	The type of attachment (also known as Internet Media Type), including text/plain, text/html, and text/xml. You can set any value here.
Destination.Email.Attachment.ContentName	The name of the attachment.
Destination.Email.Attachment.ContentEncoding	The encoding of the attachment: 7bit, base64, or quoted-printable. • 7bit is the default value that is used for ASCII text. • Base64 is used for non ASCII, whether non English or binary data. This format can be difficult to read. • Quoted-printable is an alternative to Base64, and is appropriate when the majority of the data is ASCII with some non-ASCII parts. This format is more readable; it provides a more compact encoding because the ASCII parts are not encoded.

Broker properties

You can also configure the SMTP server, port number, and security identity as a broker external resource property. To do this, use an alias that is specified in the SMTP Server and Port property on the EmailOutput node. The security identity references a user ID and password pair that is defined on the broker using the mqsisetdbparms command. Use the mqsicreateconfigurableservice command to create an SMTP broker external resource for the alias that is specified on the node. Then use the mqsichangeproperties command to create an SMTPServer property with the value in the form of *server:port*. The port value is optional; if you do not specify it, the default value is 25. You can also use the mqsichangeproperties command to create an SMTPSecurityIdentity property with a value that is the name of a security identity that can be resolved at run time to a user ID and password for authentication with the SMTP server. For example:

```
mqsicreateconfigurableservice MY_BROKER –c SMTP –o SMTP_MyAlias
```

followed by:

```
mqsichangeproperties MY_BROKER –c SMTP –o SMTP_MyAlias –n serverName –v smtp.hursley.ibm.com:25
```

These commands override the SMTP server and port values that are specified on any nodes that also specify an alias of SMTP_MyAlias. If the local environment contains any overrides, they take preference over the broker external resource properties. See also the following example:

```
mqsichangeproperties MY_BROKER –c SMTP –o SMTP_MyAlias –n securityIdentity –v mySecurityIdentity
```

You must also use the mqsisetdbparms command to define the security identity at the broker run time.

Connecting the terminals:

Connect the In terminal to the node from which outbound messages bound are routed.

Connect the Out or Failure terminal of this node to another node in this message flow to process the message further, process errors, or send the message to an additional destination.

If you connect one of these output terminals to another node in the message flow, the local environment that is associated with the message is enhanced with the following information for each destination to which the message has been put by this node:

Location	Description
WrittenDestination.Email.smtpServer	The Server:Port of the SMTP server.
WrittenDestination.Email.messageId	The ID of the e-mail sent message.

These values are written in WrittenDestination within the local environment tree structure.

If you do not connect either terminal, the local environment tree is unchanged.

Terminals and properties

The EmailOutput node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if a failure is detected when a message is propagated. Connect the Failure terminal of this node to another node in the message flow to process errors.
Out	The output terminal to which the message is routed if it has been propagated successfully. Connect the Out terminal of this node to another node in the message flow to process the message further or send the message to an additional destination.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a

value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file at deployment).

The EmailOutput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, EmailOutput	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

Use the EmailOutput node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
SMTP Server and Port	No	Yes		This property defines the SMTP server and port to which e-mails are sent from this node, and is in the format *server:port*; for example: *my.smtp.server:25*. The port value is optional, but if you do not specify a port value, the default value is 25. You can specify an alias value for this property. If the alias exists at run time, the specified values are used. If the alias does not exist at run time, the broker assumes the value to be a valid SMTP host.	smtpServer

The EmailOutput node Email properties are described in the following table.

Property	M	C	Default	Description
To Addresses	No	No		The main recipient or recipients of the e-mail. This property can include a single e-mail address or a comma-separated list of e-mail addresses.
Cc Addresses	No	No		The carbon copy recipient or recipients of the e-mail. This property can include a single e-mail address or a comma-separated list of e-mail addresses.
Bcc Addresses	No	No		The blind carbon copy recipient or recipients of the e-mail. This property can include a single e-mail address or a comma-separated list of e-mail addresses.
From Address	No	No		The e-mail address of the sender of the e-mail.
Reply-To Address	No	No		The e-mail address to which recipients of the e-mail reply.
Subject of email	No	No		The subject of the e-mail.
Email message text	No	No		The main text of the e-mail. Use this property to provide a static main body of an e-mail. If you use this property, it overrides the content that is provided in the body of the message tree that is passed to the input node. If you do not specify a value for this property, the text of the e-mail is the body of the message tree that is passed to the EmailOutput node.

Property	M	C	Default	Description
Body Content Type	No	No	text/plain	You can use this property to force the content type for the body of the e-mail message. Valid values are: • None • text/plain • text/html • text/xml

The EmailOutput node Security properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Security Identity	No	Yes		A security identifier to retrieve a user ID and password that are configured at the broker run time.	securityIdentity

The EmailOutput node Attachment properties are described in the following table.

Property	M	C	Default	Description
Attachment Content	No	No		An XPath or ESQL expression that references an element; for example, an element in the message tree, or local environment. The content of the attachment is the value of the element that is referenced.
Attachment Content Name	No	No		The name of the attachment that is seen by the recipient of the e-mail. This property is optional. If you do not specify a name, a default name is assigned.
Attachment Content Type	No	No	text/ plain	The type of the attachment. This property is optional, even if you have specified an attachment. Valid values are: • text/plain is simple text. • text/html is HTML. • text/xml is XML. • application/octet-stream is the default type for non-text and HTML (binary data).
Attachment Content Encoding	No	No	7bit	The encoding of the attachment. This property is optional. If you do not specify a value, a default encoding is assigned. Valid values are: • 7bit is the default value for ASCII text. • base64 is used for non-ASCII data, whether it is non-English or binary data. • quoted-printable is a more readable-alternative to base64. Use quoted-printable when the majority of the data is ASCII text with some non-ASCII parts. This option provides a more compact encoding because the ASCII parts are not encoded.
Multipart Content Type	No	No	Mixed	The type of multipart. Valid values are: • Mixed: each MIME body part is independent of the others. • Alternative: Each MIME body part is an alternative to the others. • Related: All MIME body parts should be considered in the aggregate only.

The Validation properties of the EmailOutput node are described in the following table.

See “Validation properties” on page 1445 for a full description of these properties.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	Yes	Yes	Inherit	This property controls whether validation takes place. Valid values are None, Content and Value, Content, and Inherit.	validateMaster
Failure action	Yes	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

EndpointLookup node

Use the EndpointLookup node to retrieve service endpoint information held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. Depending on the node configuration, the Web service URL destination used by the SOAP and HTTP request nodes can also be set. The input message is not modified.

This topic contains the following sections:

WebSphere Message Broker Version 6.1.0.4 and later supports only WSRR 6.1 or later release of version 6.

Purpose

The EndpointLookup node retrieves service endpoint information related to a WSRR service described by WSDL. A WSDL definition defines a service in terms of an interface (referred to as a portType) made available at a specified port. The WSDL port defines the endpoint information required to access the service. Endpoints are retrieved according to search criteria defined by node properties, optionally supplemented or overridden by local environment definitions at run time. See "LocalEnvironment overrides" on page 942 for more details.

The retrieved data is placed in the local environment tree, making it available to subsequent nodes. The input message received by the node is propagated to the output terminal unchanged. In addition, the EndpointLookup node can automatically set up the destination URL to be used by a subsequent SOAPRequest, SOAPAsyncRequest or HTTPRequest node, depending on the value

of the Match Policy property, see “EndpointLookup node processing.” This is done by the node setting the local environment overrides that are used by those nodes.

EndpointLookup node processing

The EndpointLookup node is contained in the **Web services** drawer of the message flow node palette, and is represented in the workbench by the following icon:

When the EndpointLookup node receives a message the following steps occur in sequence.

1. The EndpointLookup node retrieves the service data from the WSRR by using the specified search criteria.
2. If one or more matches are found, the EndpointLookup node adds a representation of those endpoints to the local environment tree.
   - If Match Policy is set to One, a single entity is returned by WSRR and added to the local environment tree. A different entity might be returned each time the query is issued. In addition, the retrieved endpoint value is set as the local environment override for the destination URL used by SOAPRequest, SOAPAsyncRequest, or HTTPRequest nodes. If the registry contains more than one entity that matches the specified search criteria it is not possible to determine which one is returned by WSRR.
   - If Match Policy is set to All, all matching entities are added to the local environment tree. The order of the entities is determined by WSRR and might vary between queries. The destination URL used by SOAPRequest, SOAPAsyncRequest, or HTTPRequest nodes is not set. Instead, you must add a compute node to your message flow to select the required address and to set up the local environment settings required by those request nodes.

   The input message is propagated unchanged to the Out terminal. The local environment tree is propagated to the Out terminal, where it is available for further processing by transformation nodes. See “EndpointLookup node output” on page 793 for details of the local environment output tree.
3. If no matches are found, the EndpointLookup node propagates the input message to the NoMatch terminal.
4. If a processing error occurs, for example if the WSRR server configured on the `DefaultWSRR` configurable service object cannot be connected to, or the connection times out, the EndpointLookup node propagates the input message unchanged to the Failure terminal. The ExceptionList is populated with details of the error.

Sample

Look at the following sample to see how to use this node:

- WSRR Connectivity

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

LocalEnvironment overrides

You can override the RegistryLookup node properties by using local environment settings. See “Dynamically defining the search criteria” on page 791.

Terminals and properties

When you have put an instance of the EndpointLookup node into a message flow, you can configure it; see “Configuring a message flow node” on page 276. The properties of the node are displayed in the Properties view.

The EndpointLookup node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if an error occurs within the node's processing.
Out	The output terminal to which the unmodified input message and updated local environment containing the matched registry data is sent.
NoMatch	The terminal to which the input message is sent if no matching entity is found based on the specified search criteria.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The EndpointLookup node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type: EndpointLookup	The name of the node.
Short description	No	No	None	A brief description of the node.
Long description	No	No	None	Text that describes the purpose of the node in the message flow.

The EndpointLookup node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
PortType Name	No	Yes	None	Name tuple that uniquely identifies a WebSphere Service Registry and Repository defined WSDL service portType. At least one of the properties is required. If you leave all three property values blank, an error message is shown when you try to save.	name
PortType Namespace	No	Yes	None		namespace
PortType Version	No	Yes	None		portVersion

Property	M	C	Default	Description	mqsiapplybaroverride command property
User Properties	No	No	None	Allows a query to specify user-defined properties. Add User Properties by clicking **Add**. User Properties refer to the Additional Properties that are used to catalogue the entities in WSRR. Enter values for Property Name, which is the case sensitive match of the additional property in WSRR, Property Type, and Property Value. The Property Type can be: • a String (the default), in which case the Property Value is a character string to be matched with the additional property value present in WSRR • XPATH, or ESQL, in which case the Property Value is a XPath or ESQL expression which locates a field in the message tree that contains the character string to be matched with the additional property value present in WSRR. These User Properties and Classification properties are used in the query to uniquely identify the WSDL service port.	
Classification	No	No	None	The Web Ontology Language (OWL) classification system property. Each classifier is a class in OWL, and has a Uniform Resource Identifier (URI). Using classifications in the registry can help to make objects easier to find and can also add meaning to custom objects that are unique to a particular system. Add a Classification by clicking **Add** and typing the complete fully-qualified OWL URI for the OWL classification. For example, it can define a particular service endpoint's lifecycle state. These User Properties and Classification properties are used in the query to uniquely identify the WSDL service port.	
Match Policy	Yes	No	One	WSRR can contain multiple entities that match the search criteria specified by the properties above. If Match Policy is set to One, at most one matching entity is returned. If Match Policy is set to All, all matching entities are returned. See “EndpointLookup node output” on page 793. If you request a single matching entity by setting Match Policy to One, the retrieved endpoint value is set as the local environment override for the destination URL used by SOAPRequest, SOAPAsyncRequest, or HTTPRequest nodes.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see “Configuring monitoring event sources using monitoring properties” on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Extract node

Use the Extract node to extract the contents of the input message that you want to be processed by later nodes in the message flow.

Attention: The Extract node is deprecated in WebSphere Message Broker Version 6.0 and later releases. Although message flows that contain an Extract node remain valid, redesign your message flows where possible to replace Extract nodes with Mapping nodes.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow"
- "Terminals and properties"

Purpose

Using the Extract node, you can create a new output message that contains only a subset of the contents of the input message. The output message comprises only those elements of the input message that you specify for inclusion when configuring the Extract node, by defining mapping statements.

The Extract node is contained in the **Database** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

You might find this node useful if you require only a subset of the message after initial processing of the whole message. For example, you might want to store the whole message for audit purposes (in the Warehouse node), but propagate only a small part of the message (order information, perhaps) for further processing.

For example, you receive orders from new clients and you want to collect their names and addresses for future promotions. Use the Extract node to get this information from each order, and send it as a new message to head office. These messages are processed at head office so that the customer details can be included in the next marketing campaign.

Terminals and properties

When you have put an instance of the Extract node into a message flow, you can configure it. For more information, see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. (If you double-click the Extract node, you open the New Message Map dialog box.) All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The Extract node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the input message is routed if a failure is detected during extraction.

Terminal	Description
Out	The output terminal to which the transformed message is routed if the input message is processed successfully.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined), the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Extract node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	Extract	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Extract node Basic properties are described in the following table.

Property	M	C	Default	Description
Mapping module	Yes	No	Extract	The name of the mapping routine that contains the statements to run against the message tree. By default, the name that is assigned to the mapping routine is identical to the name of the mappings file in which the routine is defined. The default name for the file is the name of the message flow concatenated with the name of the node when you include it in the message flow (for example, `MFlow1_Extract.msgmap` for the first Extract node in message flow MFlow1). You cannot specify a value that includes spaces. To work with the mapping routine that is associated with this node, right-click the node and click **Open Mappings**. If the mapping routine does not exist, it is created for you with the default name in the default file. If the file exists already, you can also open file *`flow_name_node_name`*`.msgmap` in the Broker Development view. A mapping routine is specific to the type of node with which it is associated; you cannot use a mapping routine that you have developed for an Extract node with any other node that uses mappings (for example, a DataInsert node). If you create a mapping routine, you cannot call it from any other mapping routine, although you can call it from an ESQL routine. For more information about working with mapping files, and defining their content, see “Developing message mappings” on page 546.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

FileInput node

Use the FileInput node to process messages that are read from files.

This topic contains the following sections:

Purpose

One or more messages can be read from a single file, and each message is propagated as a separate flow transaction. The part of a file that generates one message flow transaction is called a record. A file can be a single record, or a series of records. Properties on the node specify how the FileInput node determines the records in a file.

The FileInput node is contained in the **File** drawer of the palette, and is represented in the workbench by the following icon:

Message structure

The FileInput node handles messages in the following message domains:
- MRM
- XMLNSC
- DataObject
- XMLNS
- JMSMap
- JMSStream
- MIME
- BLOB
- XML
- IDOC

When the FileInput node propagates a message, it stores information about it in the LocalEnvironment.File message tree. If the input file is empty, an empty

message is propagated (assuming that it is valid). The following table lists the LocalEnvironment.File message tree structure. The elements contain data about the current record.

Element Name	Element Data Type	Description
Directory	CHARACTER	Absolute directory path of the input directory in the form used by the file system of the broker. For example, on Windows systems, this starts with the drive letter prefix (such as `C:`).
Name	CHARACTER	File name and extension.
LastModified	TIMESTAMP	Date and time the file was last modified.
TimeStamp	CHARACTER	Date and time, in the Coordinated Universal Time (UTC) zone, the input node started processing the file as a character string. This data is the string used to create archive and backout file names if a timestamp is included.
The following elements contain data about the current record:		
Offset	INTEGER	The start of the record within the file. The first record starts at offset 0. When Offset is part of the End of Data message tree, this value is the length of the input file.
Record	INTEGER	The number of the record within the file. The first record is record number 1. When Record is part of the End of Data message tree, this value is the number of records.
Delimiter	CHARACTER	The characters used to separate this record from the preceding record, if Delimited is specified in Record detection. The first record has a null delimiter. When Delimiter is part of the End of Data message tree, this value is the delimiter that follows the last record, if any.
IsEmpty	BOOLEAN	Whether the record that is propagated by the message flow is empty. IsEmpty is set to TRUE if the current record is empty. When IsEmpty is part of the End of Data message tree, this property is always set to TRUE.

Using this node in a message flow

The FileInput node can be used in any message flow that must accept messages in files. You can also look at the following samples to see how to use this node:

- Batch Processing
- WildcardMatch

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

If you configure a File node to use FTP, your network might need it to connect to an FTP proxy server, instead of directly to the remote FTP server. How you configure the File nodes to use an FTP proxy depends on how that proxy handles requests. For some FTP proxies, you must encode the target FTP server information in the logon credentials that you create with the mqsisetdbparms command. For example, some FTP proxies support the following values:

```
Username: FtpTargetHostUsername@ProxyUserName@TargetFtpHostname
Password: TargetFtpUserPassword@ProxyUserPassword
```

Other proxies might require different encodings, or might require external configuration, or you might not be able to use them with the File nodes.

Configuring the FileInput node

When you have put an instance of the FileInput node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (the properties that do not have a default value defined) are marked with an asterisk.

Configure the FileInput node:

1. Optional: On the **Description** tab, enter a Short description, a Long description, or both. You can also rename the node on this tab.
2. On the **Basic** tab, enter the directories and files to be processed by the FileInput node, together with what to do with any duplicate files encountered.
   - In Input directory, specify the directory from which the FileInput node obtains files. Specify the directory as either an absolute or a relative directory path. If the directory path is relative, it is based on the directory specified in the environment variable MQSI_FILENODES_ROOT_DIRECTORY. An example on Windows is `C:\fileinput`. An example on UNIX is `/var/fileinput`.

     The FileInput node creates an `mqsitransitin` subdirectory in the specified input directory. The `mqsitransitin` subdirectory holds and locks input files while they are being processed. If an execution group that processes files in this input directory is removed, remember to check the `mqsitransitin` subdirectory for partially processed or unprocessed files. Move any such files back into the input directory (and remove the execution group UUID prefix from the file names) so that they can be processed by a different execution group. For more information about the `mqsitransitin` subdirectory, see "How multiple file nodes share access to files in the same directory" on page 848.
   - In File name or pattern, specify a pattern for the file name. It is either a file name or a character sequence (a pattern) that matches a file name. A pattern is a sequence containing at least one of the following wildcard characters:

Wildcard character	Description	Example
*	Any sequence of zero or more characters	`*.xml` matches all file names with an `xml` extension
?	Any single character	`f??????.csv` matches all file names consisting of the letter f followed by six characters and then the sequence `.csv`.

     For a file to be processed, its name must match the pattern.

     If you specify a file name pattern that contains wildcard characters, the FileInput node copies the characters in the file name matched by wildcards, together with any intermediate characters, to the LocalEnvironment.Wildcard.WildcardMatch element. See "File name patterns" on page 851 for more information.
   - Select Action on successful processing to specify the action that the FileInput node takes after successfully processing the file. The action can be to move the file to the archive subdirectory, to augment the file name with a time stamp and move the source file to the archive subdirectory, or to delete the file.
     - If you select Move to Archive Subdirectory, the source file is moved to the archive subdirectory of the input directory. The subdirectory name is

mqsiarchive. For example, if the input directory is /var/fileinput, the absolute path of the archive subdirectory is /var/fileinput/mqsiarchive. If this directory does not exist, the broker creates it when it first tries to move a file there.

- If you select Add Timestamp and Move to Archive Subdirectory, the current date and time are added to the file name, and the file is then moved to mqsiarchive.
- If you select Delete, the file is deleted after successful processing.

The FileInput node writes a message to the user trace, if user tracing is in operation, whenever it processes a file.

- Select Replace duplicate archive files if you want to replace a file in the archive subdirectory with a successfully processed file of the same name. If you do not set this option, and a file with the same name exists in the archive subdirectory, the node throws an exception when it tries to move the successfully processed file.

3. On the **Input Message Parsing** tab, set values for the properties that the node uses to determine how to parse the incoming message.
   - In Message domain, select the name of the parser that you are using from the supplied list. The default is BLOB. You can choose from the following options:
     - MRM
     - XMLNSC
     - DataObject
     - XMLNS
     - JMSMap
     - JMSStream
     - MIME
     - BLOB
     - XML
     - IDOC

     You can also specify a user-defined parser, if appropriate.
   - If you are using the MRM or IDOC parser, or the XMLNSC parser in validating mode, select the Message set that you want to use. This list is populated with available message sets when you select MRM, XMLNSC, or IDOC as the domain.
   - If you are using the MRM parser, select the correct message type from the list in Message type. This list is populated with available message types when you select the MRM parser.
   - If you are using the MRM or IDOC parser, select the correct message format from the list in Message format. This list is populated with available message formats when you select the MRM or IDOC parser.
   - Specify the message coded character set ID in Message coded character set ID.
   - Select the message encoding from the list in Message encoding or specify a numeric encoding value. For more information about encoding, see "Converting data with message flows" on page 165.
4. On the **Parser Options** subtab:
   - Parse timing is, by default, set to On Demand, which causes parsing of the message to be delayed. To cause the entire message to be parsed

immediately, set this property to Immediate or Complete. See “Parsing on demand” on page 1449 for more details.

- If you are using the XMLNSC parser, set values for the properties that determine how the XMLNSC parser operates. For more information, see “Manipulating messages in the XMLNSC domain” on page 408.

5. On the **Polling** tab, enter the FileInput node's Polling interval. This property controls the frequency with which the FileInput node accesses the file system looking for files to process.

   After the initial scan of the directory when the flow is started, whenever the directory is found to contain no files that match the input pattern, the FileInput node waits for a period of time defined by this property. This process avoids the need for the FileInput node to be continually accessing the file system, and consuming large amounts of system resource.

   The smaller the value set in this property, the more quickly the FileInput node discovers files that are in the input directory. However, a smaller value increases the use of system resources. A larger value reduces the use of system resource but at the cost of the FileInput node discovering files to process less quickly.

   Do not use this property as a means to regulate work, or to schedule processing. If you want the FileInput node to monitor the input directory for selected periods only, start and stop the message flow at appropriate times.

   If you select the Remote Transfer property and set the Scan delay property on the FTP tab, the value that you set for the Scan delay overrides the value set for the Polling interval.

6. Use the **Retry** tab to define how retry processing is performed when a message flow fails:

   - Retry mechanism determines the action that occurs if the flow fails:
     - Select Failure for the node to report a failure without any retry attempts.
     - Select Short retry for the node to try again before reporting a failure if the condition persists. The number of times that it tries again is specified in Retry threshold.
     - Select Short retry and long retry for the node to try again, first using the value in Retry threshold as the number of attempts it is to make. If the condition persists after the Retry threshold has been reached, the node then uses the Long retry interval between attempts.
   - Specify the Retry threshold. The number of times the node tries the flow transaction again if the Retry mechanism property is set to either Short retry or Short retry and long retry.
   - Specify the Short retry interval. The length of time, in seconds, to wait between short retry attempts.
   - Specify the Long retry interval. The length of time to wait between long retry attempts until a message is successful, the message flow is stopped, or the message flow is redeployed. The broker property **MinLongRetryInterval** defines the minimum value that the Long retry interval can take. If the value is lower than the minimum then the broker value is used.
   - Specify the Action on failing file to determine what the node is to do with the input file after all attempts to process its contents fail:
     - Move to Backout Subdirectory. The file is moved to the backout subdirectory of the input directory. The name of this subdirectory is `mqsibackout`. If the input directory is `/var/fileinput`, the absolute path of the backout subdirectory is `/var/fileinput/mqsibackout`. If this

subdirectory does not exist, the broker creates it when it first tries to move a file there. If the file cannot be moved to this subdirectory, perhaps because a file of the same name already exists there, the node adds the current date and time to the file name and makes a second attempt to move the file. If this second attempt fails, the node stops processing, messages BIP3331 and BIP3325 are issued. Resolve the problem with the subdirectory or file before attempting to restart the message flow.

- Delete. The file is deleted after processing fails.
- Add Time Stamp and Move to Backout Subdirectory. The current date and time are added to the file name, and then the file is moved to the backout subdirectory.

7. Use the **Records and Elements** tab to specify how each file is interpreted as records:
   - Use the Record detection property to determine how the file is split into records, each of which generates a single message. Choose from the following options:
     - Whole File specifies that the whole file is a single record.
     - Fixed Length specifies that each record is a fixed number of bytes in length. Each record contains the number of bytes specified in the Length property, except possibly a shorter final record in the file.
     - Select Delimited if the records that you are processing are separated, or terminated, by a DOS or UNIX line end or by a sequence of user-defined delimiter bytes. Specify the delimiter and delimiter type in the Delimiter and Delimiter type properties.
     - Select Parsed Record Sequence if the file contains a sequence of one or more records that are serially recognized by the parser specified in Message domain. The node propagates each recognized record as a separate message. If you select the Record detection option, the parser specified in Message domain must be either XMLNSC or MRM (either CWF or TDS physical format).
   - If you specified Fixed Length in Record detection, use Length to specify the required length of the output record. This value must be between 1 byte and 100 MB. The default is 80 bytes.

     If you specify Whole File, Fixed Length, or Delimited in Record detection, a limit of 100 MB applies to the length of the records. If you specify Parsed Record Sequence in Record detection, the FileInput node does not determine or limit the length of a record. Nodes that are downstream in the message flow might try to determine the record length or process a long record. If you intend to process large records in this way, ensure that your broker has sufficient memory. You might need to apply flow techniques described in the Large Messaging sample to make best use of the available memory.
   - If you specified Delimited in Record detection, use Delimiter to specify the delimiter to be used. Choose from:
     - DOS or UNIX Line End, which, on UNIX systems, specifies the line feed character (<LF>, X'0A'), and, on Windows systems, specifies a carriage return character followed by a line feed character (<CR><LF>, X'0D0A'). The node treats both of these strings as delimiters, irrespective of the system on which the broker is running. If they are both in the same file, the node recognizes both as delimiters. The node does not recognize X'15' which, on z/OS systems, is the 'newline' byte; specify a value of Custom Delimiter in this property and a value of 15 in the Custom

delimiter property if your input file is coded using EBCDIC new lines, such as EBCDIC files from a z/OS system.
  - Custom Delimiter, which permits a sequence of bytes to be specified in Custom delimiter
- In Custom delimiter, specify the delimiter byte or bytes to be used when Custom delimiter is set in the Delimiter property. Specify this value as an even-numbered string of hexadecimal digits. The default is X'0A' and the maximum length of the string is 16 bytes (represented by 32 hexadecimal digits).
- If you specified Delimited in Record detection, use Delimiter type to specify the type of delimiter. Permitted values are:
  - Infix. If you select this value, each delimiter separates two records. If the file ends with a delimiter, the zero length file content following the final delimiter is still propagated as a message although it contains no data.
  - Postfix. If you specify this value, each delimiter terminates a record. If the file ends with a delimiter, no empty record is propagated after the delimiter. If the file does not end with a delimiter, the file is processed as if a delimiter follows the final bytes of the file. Postfix is the default value.
- The FileInput node considers each occurrence of the delimiter in the input file as either separating Infix) or terminating (Postfix) each record. If the file begins with a delimiter, the node treats the (zero length) file contents preceding that delimiter as a record and propagates an empty record to the flow. The delimiter is never included in the propagated message.

8. Use the **Validation** tab to provide validation based on the message set for predefined messages. For more information about validation, see "Validating messages" on page 204. For information about how to complete this tab, see "Validation tab properties" on page 1446.
9. On the **FTP** tab, select the Remote Transfer property if you want the node to read files from an FTP or SFTP server using the following properties:
   - In Transfer Protocol, specify the protocol that is to be used for remote file transfer. Possible values are FTP and SFTP.
   - In Remote server and port, supply the IP address and port number of the FTP or SFTP server to be used. Use the following syntax:
     - `<IP address or URL>` or
     - `<IP address or URL>:<port number>`

     If you specify the IP address in IPv6 format, ensure that you enclose it in square brackets, for example:
     - `[12a::13bd:24cd]` or
     - `[12a::13bd:24cd]:123` where 123 is the port number

     If you are using FTP and you do not specify a port number, 21 is assumed. If you are using SFTP and you do not specify a port number, a port number of 22 is assumed. However, if an FtpServer configurable service is defined, you can enter the name of the configurable service in this field. For information about how an FtpServer configurable service definition and the properties on this tab interact, see FtpServer configurable service properties.
   - In Security identity, specify the name of a security identity that has been defined using the mqsisetdbparms command. The user identifier and password that are to be used to log on to the FTP or SFTP server are obtained from this definition, the name of which must have the prefix

`ftp::`. The value of this property is overridden by the value in the securityIdentity property of the FtpServer configurable service, if it is set.

- In Server directory, specify the directory in the FTP or SFTP server from which to transfer files. The default is a period (.) which means the default directory after logon. If you specify a relative path, the directory is based on the default directory after FTP or SFTP logon. Ensure that the syntax of the path conforms to the file system standards in the FTP or SFTP server. The value in this property is overridden by the value in the remoteDirectory property of the FtpServer configurable service, if it is set.
- In Transfer mode, specify how files are transferred. Select Binary if the file contents are not to be transformed. Select ASCII if the file is to be transmitted as ASCII. The value of this property is overridden by the value in the transferMode property of the FtpServer configurable service, if it is set.

  This property is valid only when FTP is selected as the protocol for remote transfer. If you have specified SFTP as the protocol, the Transfer mode mode property is ignored and Binary encoding is used.
- In Scan delay, specify the delay, in seconds, between directory scans. The default is 60 seconds. The value set in this property overrides the value set for the polling interval on the **Polling** tab when the Remote Transfer property is selected. The value of this property is overridden by the value in the scanDelay property of the FtpServer configurable service, if it is set.

10. On the **Transactions** tab, set the transaction mode. Although all file operations are non-transactional, the transaction mode on this input node determines whether the rest of the nodes in the flow are to be executed under sync point. Select Yes if you want the flow updates to be treated transactionally, if possible, or No if you do not. The default for this property is No.
11. Optional: On the **Instances** tab, set values for the properties that control the additional instances (threads) that are available for a node. For more details, see "Configurable message flow properties" on page 1324.

Terminals and properties

The FileInput node terminals are described in the following table.

Terminal	Description
Failure	The output terminal to which a message is routed if an error occurs before a message is propagated to the Out terminal. Messages propagated to this terminal are not validated, even if you have specified, using the Validate property, that validation is to take place.
Out	The output terminal to which a message is routed if it has been successfully extracted from the input file. If no errors occur within the input node, a message received from an external resource is always sent to the Out terminal first.
End of Data	The output terminal to which the End of Data message is routed after all the messages in a file have been processed. The End of Data message flow transaction is initiated only if this terminal is attached.
Catch	The output terminal to which the message is routed if an exception is thrown downstream and caught by this node. Exceptions are caught only if this terminal is attached.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

Description properties:

Property	M	C	Default	Description
Node name	No	No	FileInput	The name of the node.
Short description	No	No	None	A brief description of the node.
Long description	No	No	None	Text that describes the purpose of the node in the message flow.

Basic properties:

Property	M	C	Default	Description	mqsiapplybaroverride command property
Input directory	Yes	Yes	None	The path of the directory from which input files are processed. The directory must be in a file system to which the broker has access. If the input directory does not exist, no files are processed. The FileInput node checks that the input directory exists at intervals that are defined by the Scan delay property. The input directory must exist, even if you are processing files over FTP or SFTP. The FileInput node creates an `mqsitransitin` subdirectory in the specified input directory. The `mqsitransitin` subdirectory holds and locks input files while they are being processed. If an execution group that processes files in this input directory is removed, check the `mqsitransitin` subdirectory for partially processed or unprocessed files. Move any such files back into the input directory (and remove the execution group UUID prefix from the file names) so that they can be processed by a different execution group.	inputDirectory
File name or pattern	Yes	Yes	*	A file name or string containing optional wildcard characters (* or ?) identifying the file or files to process from the input directory.	filenamePattern
Action on successful processing	Yes	No	Delete	The action the node takes on the file after successfully processing the contents. Valid options are: • Move to Archive Subdirectory • Add Time Stamp and Move to Archive Subdirectory • Delete	
Replace duplicate archive files	Yes	No	Cleared	This property controls whether the node replaces existing archive files with the same name as the input file. It applies only when Action on successful processing is not Delete.	

Input Message Parsing properties:

Property	M	C	Default	Description	mqsiapplybaroverride command property
Message Domain	No	No		The domain that is used to parse the incoming message.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Message Set	No	No		The name or identifier of the message set in which the incoming message is defined. If you set this property, and then update the project dependencies to remove this message set reference, a warning is issued. Either update the Message Set property, or restore the reference to this message set project.	
Message Type	No	No		The name of the incoming message.	
Message Format	No	No		The name of the physical format of the incoming message.	
Message coded character set ID	Yes	No	Broker System Default	The ID of the coded character set used to interpret bytes of the file being read.	messageCodedCharSetIdProperty
Message encoding	Yes	No	Broker System Determined	The encoding scheme for numbers and large characters used to interpret bytes of the file being read. Valid values are Broker System Determined or a numeric encoding value. For more information about encoding, see "Converting data with message flows" on page 165.	messageEncodingProperty

Parser Options properties:

Property	M	C	Default	Description
Parse timing	No	No	On Demand	This property controls when an input message is parsed. Valid values are: • On Demand • Immediate • Complete For a full description of this property, see "Parsing on demand" on page 1449.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the syntax elements in the message tree have data types taken from the XML Schema.
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	Specifies whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the message data appears under XMLNSC in nodes that are connected to the output terminal when either of the following items is XMLNS: • The input MQRFH2 header. • The Input Message Parsing property, Message Domain.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree for mixed text in an input message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.

Property	M	C	Default	Description
Retain comments	No	No	Cleared	Specifies whether the XMLNSC parser creates elements in the message tree for comments in an input message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree for processing instructions in an input message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the input message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if Validate is None); entries that are specified in Opaque Elements are ignored if validation is enabled.

Polling property:

Property	M	C	Default	Description	mqsiapplybaroverride command property
Polling interval (seconds)	Yes	Yes	5	The polling interval in seconds.	waitInterval

Retry properties:

Property	M	C	Default	Description	mqsiapplybaroverride command property
Retry mechanism	Yes	No	Failure	How the node handles a flow failure. Valid options are: • Failure • Short retry • Short and long retry	
Retry threshold	Yes	Yes	0	The number of times to try the flow transaction again when the Retry mechanism property value is Short retry.	retryThreshold
Short retry interval	No	Yes	0	The interval, in seconds, between each retry if Retry threshold property is not zero.	shortRetryInterval
Long retry interval	No	Yes	300	The interval between retries, if the Retry mechanism property is Short and long retry and the retry threshold has been exhausted.	longRetryInterval
Action on failing file	Yes	Yes	Move to Backout Subdirectory	The action that the node takes with the input file if all attempts to process the input file's contents fail. Valid options are: • Move to Backout Subdirectory • Delete • Add Time Stamp and Move to Backout Subdirectory	

Records and Elements properties:

Property	M	C	Default	Description
Record detection	Yes	No	Whole File	The mechanism used to identify records in the input file. Valid options are: • Whole File • Fixed Length • Delimited • Parsed Record Sequence
Length	Yes	No	80	The length of each record, in bytes, when Fixed Length record detection is selected.
Delimiter	Yes	No	DOS or UNIX Line End	The type of delimiter bytes that separate, or end, each record when Delimited record detection is selected. Valid options are: • DOS or UNIX Line End • Custom Delimiter
Custom delimiter	No	No		The delimiter bytes, expressed in hexadecimal, when Delimited record detection and Custom Delimiter are selected. This property is mandatory only if the Delimiter property is set to Custom Delimiter.
Delimiter type	Yes	No	Postfix	The position of the delimiter when Delimited record detection is selected. Valid options are: • Postfix • Infix This property is ignored unless the Delimiter property is set to Custom Delimiter.

Validation properties:

For a full description of these properties, see "Validation properties" on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are: • None • Content and Value • Content	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. Valid values are: • User Trace • Local Error Log • Exception • Exception List	

FTP properties:

Property	M	C	Default	Description	mqsiapplybaroverride command property
Remote Transfer	No	Yes	Cleared	This property defines whether the node uses the remote file transfer properties listed on the **FTP** tab and reads files from either an FTP or SFTP server.	fileFtp

Property	M	C	Default	Description	mqsiapplybaroverride command property
Transfer protocol	No	Yes	FTP	This property specifies the protocol to be used for remote transfer. Valid values are: • FTP • SFTP	remoteTransferType
Remote server and port	No	Yes	None	This property can have either of the following values: • The IP address or name (and, optionally, the port number) of a remote FTP or SFTP server; for example `ftp.server.com:21` or `sftp.server.com:22` • The name of a configurable service of type FtpServer If a configurable service name is specified, any or all the other remote transfer properties on the FTP tab can be overridden by the configurable service.	fileFtpServer
Security identity	No	Yes		The name of the user identification used to access the FTP or SFTP server. This property is overridden by the securityIdentity property, if set, in the FtpServer configurable service.	fileFtpUser
Server directory	No	Yes	"."	The directory on the FTP or SFTP server from which to transfer files. If you specify this property as a relative path, it is relative to the home directory after logon. This property is overridden by the remoteDirectory property, if set, in the FtpServer configurable service.	fileFtpDirectory
Transfer mode	No	No	Binary	The FTP transfer mode for transfer of file data. This property is valid only when FTP is selected as the protocol for remote transfer. Valid values are: • Binary • ASCII This property is overridden by the transferMode property, if set, in the FtpServer configurable service. If you have specified SFTP as the protocol for remote transfer, the Transfer mode property is ignored, and Binary encoding is used.	
Scan delay	No	Yes	60	The delay, in seconds, between remote directory scans. This property overrides the value set for Polling interval when the Remote Transfer property is selected. This property is overridden by the scanDelay property, if set, in the FtpServer configurable service.	

Transactions properties:

Property	M	C	Default	Description
Transaction mode	No	Yes	No	The transaction mode on this input node determines whether the rest of the nodes in the flow are executed under sync point. Valid options are: • Yes • No

Instances properties. For a full description of these properties, see "Configurable message flow properties" on page 1324.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Additional instances pool	No	Yes	Use Pool Associated with Message Flow	The pool from which additional instances are obtained. • If you select Use Pool Associated with Message Flow, additional instances are obtained from the message flow pool. • If you select Use Pool Associated with Node, additional instances are allocated from the additional instances of the node based on the number specified in the Additional instances property.	componentLevel
Additional instances	No	Yes	0	The number of additional instances that the node can start if the Additional instances pool property is set to Use Pool Associated with Node.	additionalInstances

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

FileOutput node

Use the FileOutput node to write messages to files.

This topic contains the following sections:

Purpose

You can write one or more messages from message flow transactions to a file in the broker's file system. Each message, as it is written to a file, is converted to a sequence of bytes called a *record*. Records are accumulated until a process is triggered that completes the file and places it either in the specified output

directory or a remote FTP or SFTP server directory. Properties on the node specify how records are accumulated into files and where the files are placed when they are finished.

The FileOutput node is contained in the **File** drawer of the palette and is represented in the workbench by the following icon:

Record processing

The FileOutput node writes files as a sequence of one or more records. Each record is generated from a single message received on the In terminal of the node.

By default, each file comprises a single record but properties on the FileOutput node can specify that the file comprises multiple records and how these records are accumulated in a file. Records can be accumulated in a file in the following ways:

- Concatenated: The record created from each message is appended, unmodified, to the file.
- Padded: Each record is adjusted to be a specific length and padded with a padding byte, if necessary, before being appended to the file.
- Delimited: A delimiter is used to separate or terminate the records as they are appended to the file.

For each message received, whether on the In terminal or the Finish File terminal, you can modify the output directory and the name of the file to be written (or finished) by using elements of the message. You can specify these elements, which, by default, identify elements in the local environment, on the **Request** properties tab of the node.

File processing

The FileOutput node writes accumulated messages to a file, and places it in a specified directory (the output directory) at either of the following times:

- After each record, if the file is to contain a single record. (Specify this behavior by setting the Record definition property to Record is Whole File on the **Records and Elements** tab.)
- When the Finish File terminal receives a message.

The name of the output directory and the names of the output files are determined by the node properties that you specify and by elements of the message that is being processed.

The FileOutput node uses subdirectories of the output directory to store files during and after processing. All these subdirectories begin with the prefix `mqsi`, and include subdirectories called `mqsitransit` (the transit directory) and `mqsiarchive` (the archive directory). Records are not accumulated directly into a file in the output directory but are accumulated in a file in the transit directory. Files are moved from the transit directory to the output directory when the file is complete. If a file that is to be moved to the output directory has the same name as a file that is already there, you can choose whether the file in the output directory is deleted, moved to the archive directory (`mqsiarchive`), or renamed before being moved to the archive directory.

You can specify that the FileOutput node transfers files to a remote FTP or SFTP server as part of file processing. If the file is successfully transferred, it can be deleted from the local file system, or, optionally, retained for the rest of the file processing to occur as usual.

During the file transfer operation, the FileOutput creates the destination file. However, the destination file is readable before the file transfer is complete. Therefore, ensure that remote applications do not read the file until the file transfer is complete.

When multiple records are written, no file processing occurs until a message is received on the Finish File terminal of the node. Any message received on the Finish File terminal causes the file to be moved from the transit directory to either the specified output directory or to a remote FTP or SFTP directory.

It is not an error if file processing is initiated when there is no file in the transit directory.

If you set the Record definition property to Record is Whole File on the **Records and Elements** tab, messages received on the Finish File processing are ignored because the file has already been processed.

Message propagation

For every message received on the In terminal and successfully processed by the node, a copy is propagated to the Out terminal for further processing if the terminal is attached.

For every message received on the Finish File terminal and successfully processed by the node, a copy is propagated to the End of Data terminal for further processing if the terminal is attached.

When the FileOutput node propagates a message, either to the Out terminal or to the End of Data terminal, it stores information in the LocalEnvironment.WrittenDestination.File message tree. This table describes the LocalEnvironment.WrittenDestination.File elements:

Element Name	Element Data Type	Description
Directory	CHARACTER	Absolute path of the output directory in the form used by the file system of the broker. For example, on Windows systems, the directory path starts with the drive letter prefix (such as `C:`).
Name	CHARACTER	Name of the output file.
Action	CHARACTER	Possible values are: • Replace if an output file of the same name is replaced. • Create if a new output file is created. • Append if this is associated with a record that is appended to an output file. • Finish if a Finish File message is received and no file is found to finish (for example, if Record is Whole File is specified and a message is sent to the Finish File terminal). • Transmit if the file was transferred by FTP or SFTP and the file was not retained.
Timestamp	CHARACTER	The date and time, in character string form, when the node started to process this file. This value prefixes the names of files that are archived if you set the Output file action property to Time Stamp, Archive and Replace Existing File on the **Basic** tab.

Multiple instances

Several message flows might need to write to the same file, which can happen where there are additional instances of the flow, or where multiple flows contain FileOutput nodes. The FileOutput node permits only a single instance, within an execution group and between execution groups, to write to a file at the same time. While a record is being written, all other instances in the execution group must wait. The order in which instances gain access is not defined.

When the file is complete, the first instance to gain access processes it, and other instances do not find the file. The Action element of the LocalEnvironment.WrittenDestination.File message tree is set to `Finish` for all instances that fail to discover the file in the transit directory.

Using this node in a message flow

The FileOutput node can be used in any message flow that needs to send messages to files. See "Working with files" on page 845. You can also look at the following samples to see how to use this node:

- File Output
- Batch Processing
- WildcardMatch

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

If you configure a File node to use FTP, your network might need it to connect to an FTP proxy server, instead of directly to the remote FTP server. How you configure the File nodes to use an FTP proxy depends on how that proxy handles requests. For some FTP proxies, you must encode the target FTP server information in the logon credentials that you create with the mqsisetdbparms command. For example, some FTP proxies support the following values:

```
Username: FtpTargetHostUsername@ProxyUserName@TargetFtpHostname
Password: TargetFtpUserPassword@ProxyUserPassword
```

Other proxies might require different encodings, or might require external configuration, or you might not be able to use them with the File nodes.

Configuring the FileOutput node

When you have put an instance of the FileOutput node into a message flow, you must configure it (for more information, see "Configuring a message flow node" on page 276). The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk in that view.

To configure the FileOutput node:

1. Optional: On the **Description** tab, enter a short description, a long description, or both. You can also rename the node on this tab.
2. On the **Basic** tab, enter the details of the files created by the FileOutput node.
   a. In Directory, specify the output directory in which the FileOutput node is to place its files. Specify the directory as an absolute or relative directory path. If the directory path is relative, it is based on the directory specified in the environment variable MQSI_FILENODES_ROOT_DIRECTORY. Examples

are, on Windows, `C:\fileoutput`, and, on UNIX, `/var/fileoutput`. If you want to write files in the directory which is itself identified by MQSI_FILENODES_ROOT_DIRECTORY, ensure that you specify a value of . (a period) in this property.

The output directory path to be used can be overridden by values in the current message. See the information relating to the Request tab for details about how to do this.

b. In File name or pattern, specify a file name pattern. This property defines the name of the file which is to be created by the FileOutput node. It is either a specific file name or a character sequence, a pattern, that matches a file name. Only patterns with a single wildcard character (the asterisk, '*') are allowed in this property field. The file name to be used is determined as follows:

   - If the file name contains no wildcard, the value of this property is the name of the file created. This value must be a valid file name on the file system or the FTP file system which hosts the broker to which the message flow is deployed.
   - If the file name contains a single wildcard, the value of the element LocalEnvironment.Wildcard.WildcardMatch in the current message replaces the wildcard character, and the resulting value is the name of the file created. This value must be a valid file name on the file system or the FTP file system that hosts the broker to which the message flow is deployed. If the WildcardMatch value is not found, the wildcard character is replaced by an empty string.

   The name of the file can be overridden by values in the current message. See the information relating to the Request tab for details about how to do this. If the File name or pattern property is empty, the name must be overridden by the current message. Wildcard substitution occurs only if this property is not overridden in this way.

   File names are passed to the file system to which the broker has access and have to respect the conventions of these file systems. For example, file names on Windows systems are not case-sensitive; while on UNIX systems, they are.

c. In Output file action, specify how the file is to be processed when it is complete. Choose from:

   - Replace Existing File, the default, to specify that if a file of the same name exists in the output directory, the new file replaces it.
   - Create File, to specify that a new file is created, and that if a file of the same name exists in the output directory, the new file remains in the transit directory and an exception is produced.
   - Archive and Replace Existing File, to specify that if any file of the same name exists in the output directory, it is moved to the archive directory before the new file is placed in the output directory. If any file of the same name exists in the archive directory, an exception is produced.
   - Time Stamp, Archive and Replace Existing File, to specify that if a file of the same name exists in the output directory, its name is augmented with a time stamp (a character-based version of the date and time) before being moved to the archive directory.

d. Select the Replace duplicate archive files check box to specify that, in cases where Archive and Replace Existing File or Time Stamp, Archive and Replace Existing File is specified in Output file action, files moved to the archive directory replace files that exist there already with the same name. By default, this check box is not selected. If this check box is not selected,

and there is already a file in the archive directory with the same name as a file that is to be moved there, an exception is produced, and the new file remains in the transit directory.

3. On the **Request** tab, specify the location of the data to be written, and control information that overrides the **Basic** tab's Directory and File name or pattern properties. You can specify the properties on this tab as XPath or ESQL expressions. Content-assist is available in the properties pane and also in the XPath Expression Builder, which you can invoke by using the **Edit** button to the right of each property.
   a. In Data location, specify the input data location. This is the location in the input message tree that contains the record to be written to the output file. The default value is $Body, meaning the entire message body ($InputRoot.Body).

      When you specify this property, and the data in the message tree that it identifies is owned by a model-driven parser, such as the MRM parser or XMLNSC parser, consider the following:

      - If you are using MRM CWF format, ensure that the identified message tree exists as a message definition. If this is defined as a global element only, exceptions BIP5180 and BIP5167 are produced.
      - If you are using MRM TDS format, serialization of the identified message is successful if the element is defined as a global element or message. However, if the identified field is not found as a global element or message, note that:
        – If it is a leaf field in the message tree, the field is written as self-defining. No validation occurs even if validation is enabled.
        – If it is a complex element, an internal exception is generated, BIP5522, indicating that the logical type cannot be converted to a string.
      - If you are using MRM XML, the events are similar as for the MRM TDS format except that, if the field is a complex element, it is written as self-defining.
      - If you use the XMLNSC parser, no validation occurs even if validation is enabled.

   b. In Request directory property location, specify the location of the value to override the Directory property on the **Basic** tab. If you do not specify a location, the default value is $LocalEnvironment/Destination/File/Directory. If you specify a location but the element is empty or missing, the Directory property is used. The element is defined as follows:

Element data type	Element attributes
CHARACTER	Absolute or relative directory path. Use the path separator character ('/' or '\') according to the file system on which the broker is executing. Trailing path separator characters are ignored. Relative directory paths are based on the value of the MQSI_FILENODES_ROOT_DIRECTORY environment variable.

   c. In Request file name property location, specify the location of the value to override the File name or pattern property on the **Basic** tab. If you do not specify a location, the default value is $LocalEnvironment/Destination/File/Name. If you specify a location but the element is empty or missing, the File name or pattern property is used. The element is defined as follows:

Element data type	Element attributes
CHARACTER	Explicit file name. No wildcard substitution occurs for this value.

4. Use the **Records and Elements** tab to specify how the FileOutput node writes the record derived from the message.
   - In Record definition, choose from:
     - Record is Whole File to specify that the file is to contain a single record. The file is finished immediately after the record is written; the FileOutput node does not wait for a message on the Finish File terminal. This is the default.
     - Record is Unmodified Data to specify that records are accumulated in a file with neither padding or delimiters applied. The file is finished only when a message is received on the Finish File terminal.
     - Record is Fixed Length Data to specify that records are padded to a given length if necessary and accumulated in a file by concatenation. You specify this length in the Length property. If the record is longer than the value specified in Length, the node produces an exception. Use the Padding byte property to specify the byte to be used for padding the message to the required length. Records are added to this file until a message is received on the Finish File terminal.
     - Record is Delimited Data to specify that records are separated by a delimiter and accumulated by concatenation. The delimiter is specified by the Delimiter, Custom delimiter, and Delimiter type properties. Records are added to this file until a message is received on the Finish File terminal.
   - In Length, specify the length (in bytes) of records when Record is Fixed Length Data is specified in Record definition. Records longer than this value cause an exception to be produced. This value must be between 1 byte and 104857600 bytes (100 MB). The default is 80 bytes.
   - When Record is Fixed Length Data is specified in Record definition, use Padding byte to specify the byte to be used when padding records to the specified length if they are shorter than this length. Specify this value as two hexadecimal digits. The default value is X'20'.
   - In Delimiter, specify the delimiter to be used if you specify Record is Delimited Data in Record definition. Choose from:
     - Broker System Line End to specify that a line end sequence of bytes is used as the delimiter as appropriate for the file system on which the broker is to run. This is the default. For example, on Windows systems, it is a 'carriage-return, line-feed' pair (X'0D0A'); on UNIX systems, it is a single 'line-feed' byte (X'0A'); on z/OS systems, it is a 'newline' byte (X'15').
     - Custom Delimiter to specify that the explicit delimiter sequence defined in the Custom delimiter property is to be used to delimit records.
   - In Custom delimiter, specify the delimiter sequence of bytes to be used to delimit records when Custom Delimiter is specified in the Delimiter property. Specify this value as an even-numbered string of hexadecimal digits. The default is X'0A' and the maximum length of the string is 16 bytes.
   - If you specified Record is Delimited Data in Record definition, use Delimiter type to specify how the delimiter is to separate records. Choose from:
     - Postfix to specify that the delimiter is added after each record that is written. This is the default.

- Infix to specify that the delimiter is inserted only between any two adjacent records.

5. On the **Validation** tab, specify the parser validation properties of the node. For more information about validation, see "Validating messages" on page 204. For information about how to complete this tab, see "Validation properties" on page 1445.
6. On the **FTP** tab, select the Remote Transfer property if you want the node to transfer files to an FTP or SFTP server using the following properties:
   - In Transfer Protocol, specify the protocol that is to be used for remote file transfer. Possible values are FTP and SFTP.
   - In Remote server and port, supply the IP address and port number of the FTP or SFTP server to be used. Use the following syntax:
     - *IP_address_or_URL* or
     - *IP_address_or_URL:port_number*

     If you specify the IP address in IPv6 format, ensure that you enclose it in square brackets, for example:
     - `[12a::13bd:24cd]` or
     - `[12a::13bd:24cd]:123` where `123` is the port number

     If you are using FTP and you do not specify a port number, 21 is assumed. If you are using SFTP and you do not specify a port number, a port number of 22 is assumed. However, if an FtpServer configurable service is defined, you can enter the name of the configurable service in this field. For information about how an FtpServer configurable service definition and the properties on this tab interact, see FtpServer configurable service properties.
   - In Security identity, specify the name of a security identity that has been defined using the mqsisetdbparms command. The user identifier and password that are to be used to log on to the FTP or SFTP server are obtained from this definition, the name of which must have the prefix `ftp::`. The value of this property is overridden by the value in the FtpServer configurable service property securityIdentity, if it is set.
   - In Server directory, specify the directory in the FTP or SFTP server to which to transfer files. The default is `.` (a period) which means the default directory after login. If you specify a relative path, the directory is based on the default directory after FTP or SFTP logon. Ensure that the syntax of the path conforms to the file system standards in the FTP or SFTP server. The value of this property is overridden by the value in the remoteDirectory property of the FtpServer configurable service, if it is set.
   - In Transfer mode, specify how files are transferred. Select Binary if the file contents are not to be transformed. Select ASCII if the file is to be transmitted as ASCII. The value of this property is overridden by the value in the FtpServer configurable service property transferMode, if it is set.

     This property is valid only when FTP is selected as the protocol for remote transfer. If you have specified SFTP as the protocol, the Transfer mode mode property is ignored and Binary encoding is used.
   - Select the Retain local file after transfer check box if you want to retain a local copy of the file after the file transfer process has completed. If this check box is selected, the local copies are processed after transfer as are other output files, as specified on the **Basic** tab. If it is not selected, successfully transferred files are not retained locally.

Terminals and properties

The FileOutput node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Finish File	The input terminal that accepts a message that triggers the final processing of a file.
Out	The message received on the In terminal is propagated to this terminal if the record is written successfully. The message is unchanged except for status information in the Local Environment.
End of Data	The message received on the Finish File terminal is propagated to this terminal if the file is processed successfully.
Failure	The output terminal to which the message is routed if a failure is detected when a message is propagated.

The following tables describe the node properties that you can set on a specified tab. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

Description properties

Property	M	C	Default	Description
Node name	No	No	FileOutput	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

Basic properties

Property	M	C	Default	Description	mqsiapplybaroverride command property
Directory	No	Yes	None	The output directory where the node places its files.	outputDirectory
File name or pattern	No	Yes	None	The specific file name or a pattern containing a single wildcard which defines the name of the file to be created.	outputFilename
Output file action	Yes	No	Replace Existing File	Specifies the action to be taken when the output file is finished. Valid options are: • Replace Existing File • Create File • Archive and Replace Existing File • Time Stamp, Archive and Replace Existing File	
Replace duplicate archive files	Yes	No	Cleared	Specifies whether files in the archive directory can be replaced by files of the same name being moved there.	

Request properties

Property	M	C	Default	Description	mqsiapplybaroverride command property
Data location	Yes	No	$Body	The location in the input message tree containing the record written to the output file.	
Request directory property location	Yes	Yes	$LocalEnvironment/Destination/File/Directory	The message element location containing the name of the output directory.	requestDirectoryLocation
Request file name property location	Yes	Yes	$LocalEnvironment/Destination/File/Name	The message element location containing the name of the output file.	requestNameLocation

Records and Elements properties

Property	M	C	Default	Description
Record definition	Yes	No	Record is Whole File	This property controls how the records are placed in the output file. Valid options are: • Record is Whole File • Record is Unmodified Data • Record is Fixed Length Data • Record is Delimited Data
Length	Yes	No	80	The required length of the output record. This property applies only when Record is Fixed Length Data is specified in Record definition.
Padding byte	Yes	No	X'20'	The 2-digit hexadecimal byte to be used to pad short messages when Record is Fixed Length Data is specified in Record definition.
Delimiter	Yes	No	Broker System Line End	The delimiter to be used when Record is Delimited Data is specified in Record definition. Valid options are: • Broker System Line End • Custom Delimiter
Custom delimiter	No	No	None	The delimiter byte sequence to be used when Record is Delimited Data is specified in the Record definition property and Custom Delimiter is specified in the Delimiter property.
Delimiter type	Yes	No	Postfix	This property specifies the way in which the delimiters are to be inserted between records when Record is Delimited Data is specified in Record definition. Valid options are: • Postfix • Infix

Validation properties

For a full description of these properties, see “Validation properties” on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	Inherit	This property controls whether validation takes place. Valid values are: • None • Content and Value • Content • Inherit	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are: • User Trace • Local Error Log • Exception • Exception List	

FTP properties

Property	M	C	Default	Description	mqsiapplybaroverride command property
Remote Transfer	No	Yes	Cleared	This property defines whether the node uses the remote file transfer properties listed on the **FTP** tab and transfers files to an FTP or SFTP server.	fileFtp
Transfer protocol	No	Yes	FTP	This property specifies the protocol to be used for remote transfer. Valid values are: • FTP • SFTP	remoteTransferType
Remote server and port	No	Yes	None	This property can have either of the following values: • The IP address or name (and, optionally, the port number) of a remote FTP or SFTP server; for example `ftp.server.com:21` or `sftp.server.com:22` • The name of a configurable service of type FtpServer If a configurable service name is specified, any or all of the other remote transfer properties on the FTP tab can be overridden by the configurable service.	fileFtpServer
Security identity	No	Yes	None	The name of the user identification used to access the FTP or SFTP server. This property is overridden by the securityIdentity property, if set, in the FtpServer configurable service.	fileFtpUser
Server directory	No	Yes	"."	The directory on the FTP or SFTP server to which to transfer files. If you specify this property as a relative path, it is relative to the home directory after logon. This property is overridden by the remoteDirectory property, if set, in the FtpServer configurable service.	fileFtpDirectory

Property	M	C	Default	Description	mqsiapplybaroverride command property
Transfer mode	No	Yes	Binary	The FTP transfer mode for transfer of file data. This property is valid only when FTP is selected as the protocol for remote transfer. Valid options are: • Binary • ASCII This property is overridden by the transferMode property, if set, in the FtpServer configurable service. If you have specified SFTP as the protocol for remote transfer, the Transfer mode mode property is ignored, and Binary encoding is used.	
Retain local file after transfer	No	No	Cleared	If this property is selected, a local copy of a file that has been transferred to a remote FTP or SFTP server is retained and placed in the output directory in accordance with the disposition of files as specified on the **Basic** tab. If this property is not selected, local copies of files that have been transferred to a remote FTP or SFTP server are not retained.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Filter node

Use the Filter node to route a message according to message content.

This topic contains the following sections:

Purpose

Create a filter expression in ESQL to define the route that the message is to take. You can include elements of the input message or message properties in the filter expression, and you can use data that is held in an external database to complete the expression. The output terminal to which the message is routed depends on whether the expression evaluates to true, false, or unknown.

Connect the terminals that cover all situations that could result from the filter; if the node propagates the message to a terminal that is not connected, the message is discarded even if it is transactional.

The Filter node accepts ESQL statements in the same way as the Compute and Database nodes. The last statement that is executed must be a `RETURN <expression>`

statement, whose expression evaluates to a Boolean value. This Boolean value determines the terminal to which the message is routed. In many cases, the routing algorithm is a simple comparison of message field values. The comparison is described by the expression and the RETURN statement is the only statement. If you code RETURN without an expression (`RETURN;`) or with a null expression, the node propagates the message to the Unknown terminal.

If your message flow requires more complex routing options, use the RouteToLabel and Label nodes.

The Filter node is contained in the **Routing** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Look at the following samples for examples of how to use this node:
- Airline Reservations
- Scribble
- Error Handler
- Large Messaging

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

Consider a situation in which you have produced an online test with ten multiple choice questions. Each message coming in has a candidate name and address followed by a series of answers. Each answer is checked, and if it is correct, the field SCORE is incremented by one. When all the answers have been checked, the field SCORE is tested to see if it is greater than five. If it is, the Filter node propagates the message to the flow that handles successful candidate input; otherwise, the message is filtered into the rejection process, and a rejection message is created.

Terminals and properties

When you have put an instance of the Filter node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The Filter node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node
Failure	The output terminal to which the message is routed if a failure is detected during the computation
Unknown	The output terminal to which the message is routed if the specified filter expression evaluates to `unknown` or a null value
False	The output terminal to which the message is routed if the specified filter expression evaluates to `false`
True	The output terminal to which the message is routed if the specified filter expression evaluates to `true`

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default value is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Filter node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short Description	No	No		A brief description of the node
Long Description	No	No		Text that describes the purpose of the node in the message flow

The Filter node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Data Source	No	Yes		The ODBC data source name of the database that contains the tables to which you refer in the ESQL that is associated with this node (identified by the Filter Expression property). This name identifies the appropriate database on the system on which this message flow is to execute. The broker connects to this database with user ID and password information that you have specified on the mqsicreatebroker, mqsichangebroker, or mqsisetdbparms command. z/OS On z/OS systems, the broker uses the broker started task ID, or the user ID and password that are specified on the mqsisetdbparms command JCL, BIPSDBP, in the customization data set <hlq>.SBIPPROC. If the ESQL that is associated with this node includes a PASSTHRU statement or SELECT function and a database reference, you must specify a value for the Data Source property.	dataSource
Transaction	Yes	No	Automatic	The transaction mode for the node. The values are: • Automatic (the default). The message flow, of which the Filter node is a part, is committed if it is successful. That is, the actions that you define in the ESQL module are performed and the message continues through the message flow. If the message flow fails, it is rolled back. Therefore, if you choose Automatic, the ability to commit or roll back the action of the Filter node on the database depends on the success or failure of the entire message flow. • Commit. To commit any uncommitted actions that are performed in this message flow on the database that is connected to this node, irrespective of the success or failure of the message flow as a whole, select Commit. The changes to the database are committed even if the message flow itself fails.	

Property	M	C	Default	Description	mqsiapplybaroverride command property

Filter Expression	Yes	No	Filter	The name of the module within the ESQL resource (file) that contains the statements to execute against the message that is received in the node The ESQL file, which by default has the name `<message_flow_name>.esql`, contains ESQL for every node in the message flow that requires it. Each portion of code that is related to a specific node is known as a module. If you want the module name to include one or more spaces, enclose it in double quotation marks in the Filter Expression property.
				
				Code ESQL statements to customize the behavior of the Filter node in an ESQL file that is associated with the message flow in which you have included this instance of the Filter node.
				
				If an ESQL file does not already exist for this message flow, double-click the Filter node, or right-click the node and click **Open ESQL** to create and open a new ESQL file in the ESQL editor view.
				
				If the file exists already, click **Browse** beside the Filter Expression property to display the Module Selection dialog box, which lists the available Filter node modules defined in the ESQL files that can be accessed by this message flow (ESQL files can be defined in other, dependent, projects). Select the appropriate module and click **OK**; if no suitable modules are available, the list is empty.
				
				If the module that you specify does not exist, that module is created for you, and the editor displays it. If the file and the module exist already, the editor highlights the correct module.
				
				If a module skeleton is created for this node in a new or existing ESQL file, it consists of the following ESQL. The default module name is shown in this example:
				`CREATE FILTER MODULE <flow_name>_Filter`
				`  CREATE FUNCTION Main() RETURNS BOOLEAN`
				`  BEGIN`
				`  RETURN TRUE;`
				`  END;`
				`END MODULE;`
				
				If you create your own ESQL module, you must create this skeleton exactly. You can update the default name, but ensure that the name that you specify matches the name of the corresponding node property Filter Expression.
				
				To customize this node, add your own ESQL after the BEGIN statement, and before the RETURN statement. If the expression on the RETURN statement is not TRUE or FALSE, its value is resolved to determine the terminal to which the message is propagated. If the expression resolves to a null value, or you code `RETURN;`, or you omit the RETURN statement, the node propagates the message to the Unknown terminal.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
				You can use all the ESQL statements including SET, WHILE, DECLARE, and IF in this module, but (unlike the Compute node) the Filter node propagates the message that it receives at its input terminal to its output terminal unchanged. Therefore, in the Filter node, like the Database node, you have only one message to which to refer. The ESQL correlation names that you use in a Filter node are different from those used for a Compute node. For more information about correlation names refer to the related links. You cannot modify any part of any message, so the assignment statement (the SET statement, not the SET clause of the INSERT statement) can assign values only to temporary variables. The scope of actions that you can take with an assignment statement is therefore limited.	
Treat warnings as errors	Yes	No	Cleared	For database warning messages to be treated as errors, and to propagate the output message from the node to the Failure terminal, select Treat warnings as errors. The check box is cleared initially. When you select the check box, the node handles all positive return codes from the database as errors and generates exceptions in the same way as it does for the negative, or more serious, errors. If you do not select the check box, the node treats warnings as normal return codes and does not raise any exceptions. The most significant warning raised is `not found`, which can be handled safely as a normal return code in most circumstances.	
Throw exception on database error	Yes	No	Selected	For the broker to generate an exception when a database error is detected, select Throw exception on database error. The check box is selected initially. If you clear the check box, you must include ESQL to check for any database error that might be returned after each database call that you make (you can use SQLCODE and SQLSTATE to do this). If an error has occurred, you must handle the error in the message flow to ensure the integrity of the broker and the database; the error is ignored if you do not handle it through your own processing because you have chosen not to invoke the default error handling by the broker. For example, you can include the ESQL THROW statement to throw an exception in this node, or you can use the Throw node to generate your own exception at a later point.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

FlowOrder node

Use the FlowOrder node to control the order in which a message is processed by a message flow.

This topic contains the following sections:

Purpose

The FlowOrder node propagates the input message to the first output terminal, and the sequence of nodes that is connected to this terminal processes the message. When that message processing is complete, control returns to the FlowOrder node. If the message processing completes successfully, the FlowOrder node propagates the input message to the second output terminal, and the sequence of nodes that is connected to this terminal processes the message.

The message that is propagated through the second output terminal is the input message; it is not modified in any way, even if the sequence of nodes that is connected to the first terminal has modified the message.

You can include this node in a message flow at any point where the order of execution of subsequent nodes is important.

If you connect multiple nodes to the first output terminal, the second output terminal, or both, the order in which the multiple connections on each terminal are processed is random and unpredictable. However, the message is propagated to all target nodes that are connected to the first output terminal, which must all complete successfully, before the message is propagated to any node that is connected to the second output terminal.

Your message flow performance can benefit from including the FlowOrder node in a situation where one sequence of processing that is required for a message is significantly shorter than another sequence of processing. If you connect the shorter sequence to the first terminal, any failure is identified quickly and prevents execution of the second longer sequence of processing.

The FlowOrder node is contained in the **Construction** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

For an example of using this node, assume that your company receives orders from customers using the Internet. When the order is received, it is processed by nodes that are connected to the first terminal of a FlowOrder node to debit the stock level in your database and raise an invoice. A check is made to see whether the customer has indicated that his details can be sent to other suppliers. If the customer has indicated that he does not want this information to be divulged, this check fails and no further processing occurs. If the customer is happy for you to share his details with other companies (that is, the test is successful), the input message is propagated to the second terminal so that the customer's details can be added to the mailing list.

Connecting the terminals

The FlowOrder node has no configurable properties that affects its operation. You determine how it operates by connecting the first and second output terminals to subsequent nodes in your message flow.

1. Connect the First terminal to the first node in the sequence of nodes that provide the first phase of processing this message. This sequence can contain one or more nodes that perform any valid processing. The sequence of nodes can optionally conclude with an output node.
2. Connect the Second terminal to the first node in the sequence of nodes that provide the second phase of processing this message. This sequence can contain one or more nodes that perform any valid processing. The sequence of nodes can optionally conclude with an output node.

   The message that is propagated through the Second terminal is identical to that propagated through the First terminal. Any changes that you have introduced as a result of the first phase of processing are ignored by this node.

   If the first phase of processing fails, the FlowOrder node does not regain control and does not propagate the message through the Second terminal.

Terminals and properties

When you have put an instance of the FlowOrder node into a message flow, you can configure it. For more information, see “Configuring a message flow node” on page 276. The properties of the node are displayed in the Properties view.

The FlowOrder node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if a failure is detected during the computation.
First	The output terminal to which the input message is routed in the first instance.
Second	The output terminal to which the input message is routed in the second instance. The message is routed to this terminal only if routing to First is successful.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The FlowOrder node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	FlowOrder	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

HTTPHeader node

Use the HTTPHeader node to add, modify, or delete HTTP headers such as HTTPInput, HTTPResponse, HTTPRequest and HTTPReply.

This topic contains the following sections:

Purpose

The HTTPHeader node provides a toolkit interface to manipulate HTTP headers without any need for coding; it does not modify the message body. You can add or remove a whole header, or selected header properties. You can set the properties to a fixed value, or to a value specified by an XPath expression that accesses a value in one of the message trees. XPath is used to provide a valid location from which a value for a property can be copied. For example, the location can be the body of the message, the local environment tree or exception list.

HTTPInput and HTTPResponse headers can only be deleted or carried forward from the incoming message; their header properties cannot be modified or added to.

The HTTPHeader node is contained in the **HTTP** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Look at the following sample for more details about how to use the node:

- HTTPHeader node

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

Terminals and properties

When you have put an instance of the node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. This node has no mandatory properties.

HTTPHeader node terminals are described in the following table:

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the input message is routed if a failure is detected during extraction.
Out	The output terminal to which the transformed message is routed if the input message is processed successfully.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The HTTPHeader node Description properties are described in the following table:

Property	M	C	Default	Description
Node name	No	No	HTTPHeader	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

HTTPInput Header properties are described in the following table:

Property	M	C	Default	Description
HTTPInput Header Options	No	Yes	Carry forward header	Options to control the HTTPInputHeader as a whole. Select **Carry forward the header** to carry forward values from incoming message if present. Select **Delete header** to delete the header if present.

HTTPResponse Header properties are described in the following table:

Property	M	C	Default	Description
HTTPResponse Header Options	No	Yes	Carry forward header	Options to control the HTTPResponseHeader as a whole. Select **Carry forward the header** to carry forward values from incoming message if present. Select **Delete header** to delete the header if present.

HTTPRequest Header properties are described in the following table:

Property	M	C	Default	Description
HTTPRequest Header Options	No	Yes	Carry forward header	Configure the HTTPRequest header. These options are available. **Carry forward header** Select this option to carry forward values from an incoming message. **Add header** Select this option to add new properties to the header, or to modify or delete existing properties. **Modify header** Select this option to add properties, or modify and delete existing properties. **Delete header** Select this option to remove the HTTPRequest header and all associated properties from the incoming message.
Clear incoming values	No	Yes	Cleared	This option, which is enabled only if you choose **Modify header**, removes all property names and associated values from the incoming message if present.

Property	M	C	Default	Description
HTTPRequest Header	No	Yes	No default value	This field is enabled only if you chose **Add header** or **Modify header** for the **HTTPRequest Header Options**. The screen has no predefined properties; you use it to create custom properties and values. Use the property table to add new properties, or modify or delete existing properties, for the header. There is no limit to the number of properties. Each property must have a name and a type qualifier. The type qualifier can be `Value`, `XPath`, or `Delete`. **Value** Enter a new valid value for the selected property. A null value or empty string is also considered as a valid value. **XPath** Specify a valid XPath expression. WebSphere Message Broker supports XPath definitions that start with an XPath variable such as `$Root` or `$LocalEnvironment`. Only the first occurrence is returned if there are multiple values for the XPath expression. (Examples of valid XPath expressions are: $LocalEnvironment/Host, and $Root/HTTPRequest/Content-Type). **Delete** Specify the property to be deleted from the incoming message. The value associated with the selected property is also deleted.

HTTPReply Header properties are described in the following table:

Property	M	C	Default	Description
HTTPReply Header Options	No	Yes	Carry forward header	Configure the HTTPReply header. These options are available. **Carry forward header** Select this option to carry forward values from an incoming message. **Add header** Select this option to add new properties to the header, or to modify or delete existing properties. **Modify header** Select this option to add properties, or modify and delete existing properties. **Delete header** Select this option to remove the HTTPReply header and all associated properties from the incoming message.
Clear incoming values	No	Yes	Cleared	This option, which is is enabled only if you choose **Modify header**, removes all property names and associated values from the incoming message if present.

Property	M	C	Default	Description
HTTPReply Header	No	Yes	No default value	This field is enabled only if you chose **Add header** or **Modify header** for **HTTPRequest Header Options**. The screen has no predefined properties; you use it to create custom properties and values. Use the property table to add new properties, or modify or delete existing properties, for the header. There is no limit to the number of properties. Each property must have a name and a type qualifier. The type qualifier can be `Value`, `XPath`, or `Delete`. **Value** Enter a new valid value for the selected property. A null value or empty string is also considered as a valid value. **XPath** Specify a valid XPath expression. WebSphere Message Broker supports XPath definitions that start with an XPath variable such as `$Root` or `$LocalEnvironment`. Only the first occurrence is returned if there are multiple values for the XPath expression. (Examples of valid XPath expressions are: $LocalEnvironment/Host, and $Root/HTTPRequest/Content-Type). **Delete** Specify the property to be deleted from the incoming message. The value associated with the selected property is also deleted.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

HTTPInput node

Use the HTTPInput node to receive an HTTP message from an HTTP client for processing by a message flow.

This topic contains the following sections:

Purpose

If you use the HTTPInput node with the HTTPReply and HTTPRequest nodes, the broker can act as an intermediary for Web services, and Web service requests can

be transformed and routed in the same way as other message formats that are supported by WebSphere Message Broker.

Web service requests can be received either in standard HTTP (1.0 or 1.1) format or in HTTP over SSL (HTTPS) format. For more information about Web services, see “Working with Web services” on page 735.

The HTTPInput node supports HTTP POST and HTTP GET. For more information about enabling HTTP GET, see “HTTPRequest node” on page 991.

If your message flows are processing SOAP messages, use the SOAP nodes in preference to the HTTPInput node to take advantage of enhanced features, including WS-Addressing and WS-Security.

The HTTPInput node handles messages in the following message domains:
- MRM
- XMLNSC
- XMLNS
- MIME
- BLOB
- XML (this domain is deprecated; use XMLNSC)

HTTP messages are always non-persistent, and have no associated order.

HTTP messages are non-transactional. However, if the message flow interacts with a database or another external resource, such as a WebSphere MQ queue, these interactions are performed in a transaction. The HTTPInput node provides commit or rollback, depending on how the message flow has ended, and how it is configured for error handling (how failure terminals are connected, for example). If the message flow is rolled back by this node, a fault message is generated and returned to the client. The format of the fault is defined by the Fault format property.

If an exception occurs downstream in this message flow, and it is not caught but is returned to this node, the node constructs an error reply to the client. This error is derived from the exception, and the format of the error is defined by the Fault format property.

If you include an output node in a message flow that starts with an HTTPInput node, the output node can be any of the supported output nodes (including user-defined output nodes). You can create a message flow that receives messages from Web service clients, and generates messages for clients that use all the supported transports to connect to the broker. You can configure the message flow to request the broker to provide any conversion that is necessary.

If you create a message flow to use as a subflow, you cannot use a standard input node; you must use an Input node as the first node to create an In terminal for the subflow.

The HTTPInput node is contained in the **HTTP** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

If you include an HTTPInput node in a message flow, you must either include an HTTPReply node in the same flow, or pass the message to another flow that includes an HTTPReply node (for example, through an MQOutput node to a second flow that starts with an MQInput node). In the latter case, the request from, and reply to, the client are coordinated by the request identifier stored in the local environment by the HTTPInput node.

When the HTTPInput node receives a message from a Web service client, the node starts the appropriate parsers to interpret the headers and the body of the message, and to create the message tree that is used internally by the message flow. The node creates a unique identifier for the input message and stores it as a binary array of 24 bytes in the local environment tree at LocalEnvironment.Destination.HTTP.RequestIdentifer. This value is used by the HTTPReply node, therefore you must not modify it.

Using the HTTPInput and HTTPReply nodes to act as a Web server:

A broker can support multiple HTTPInput nodes. When you configure the HTTPInput node, specify the requests to which the node listens in the form of a URL path, excluding the host and port details.

For example, if the broker is listening on address `http://localhost:7080`, and receives the request `http://localhost:7080/Joe/Mary`, the listener removes the HTTP address, leaving the request `Joe/Mary`. The listener then matches this request with the information that is specified in the URL property of the HTTPInput node or nodes.

The match is done from the most specific to the most generic data; you can use a wildcard (an asterisk) to satisfy less specific matches. For example, if you have configured an HTTPInput node to accept requests that match `/Joe/Mary`, that node receives the message. However if the request is `http://localhost:7080/Joe/Sally`, the match is not made with this node. It can match with a node that has a more generic URL, such as one of the following values:

```
/Joe/*
/*
```

If the request does not match any URL property, and you do not have an input node with /* specified, the HTTPInput node returns a response to the originator.

You can use a URL of /* to catch all requests that failed to match the URLs in the HTTPInput nodes, so that you can send a reply message and take other actions as appropriate.

If you choose to handle HTTP messages by using the listener that is embedded in the execution group, which is also used for SOAP nodes, you must carefully check the URL specifications in your HTTPInput and SOAPInput nodes. If you use wildcards in the Path suffix for URL property of the HTTPInput node, or the URL Selector property of the SOAPInput node, check that the URL cannot match both HTTP and SOAP nodes. If both properties match an incoming message, the listener might route the message to the wrong type of node, and processing might fail or produce unexpected results.

You must configure the broker properties for the listener to define a port for HTTP and HTTPS messages. The default port for HTTP is 7080, no default port for

HTTPS is configured. The HTTP listener is started by the broker when a message flow that includes HTTP nodes or Web services support is started. When a request comes in, the listener creates a WebSphere MQ message with the input message, and puts the request to a WebSphere MQ, from where the HTTPInput retrieves it.

The HTTP listener listens on Internet Protocol Version 6 (IPv6) and Internet Protocol Version 4 (IPv4) addresses where supported by the operating system. To enable IPv6 listening on Windows and HP-UX platforms, set the environment variable _JAVA_OPTIONS to `Djava.net.preferIPv4Stack=false`.

You can use the mqsichangetrace command to collect trace information for the HTTP listener. To process the log, use the mqsireadlog command with **Qualifier** set to httplistener.

Connecting the terminals

The HTTPInput node routes each message that it retrieves successfully to the Out terminal. If message validation fails, the message is routed to the Failure terminal; you can connect nodes to this terminal to handle this condition. If you have not connected the Failure terminal, the message is discarded, the Maximum client wait time expires, and an error is returned to the client. No other situations exist in which the message is routed to the Failure terminal.

If the message is caught by this node after an exception has been thrown further on in the message flow, the message is routed to the Catch terminal. If you have not connected the Catch terminal, the message is discarded, the Maximum client wait time expires, and an error is returned to the client.

Terminals and properties

When you have put an instance of the HTTPInput node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view.

The terminals of the HTTPInput node are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is routed if an error occurs.
Out	The output terminal to which the message is routed if it is successfully retrieved.
Catch	The output terminal to which the message is routed if an exception is thrown downstream and caught by this node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The HTTPInput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, HTTPInput	The name of the node.

Property	M	C	Default	Description
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The HTTPInput node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Path suffix for URL	Yes	Yes		This property identifies the location from where Web service requests are retrieved. Do not use the full URL. If the URL that you want is `http://hostname[:port]/[path]`, specify either `/path` or `/path fragment/*` where * is a wildcard that you can use to mean match any.	URLSpecifier
Use HTTPS	No	Yes	Cleared	This property identifies whether the node is to accept secure HTTP. If the node is to accept secure HTTP, select the check box.	useHTTPS

The HTTPInput node Advanced properties are described in the following table.

Property	M	C	Default	Description
Set destination list	No	No	Selected	This property specifies whether to add the method binding name to the route to label destination list. If you select this check box, the method binding name is added so that you can use a RouteToLabel node in the message flow after the HTTPInput node.
Label prefix	No	No	None	The prefix to add to the method name when routing to label. Add a label prefix to avoid a clash of corresponding label nodes when you include multiple WebSphere Message Broker input nodes in the same message flow. By default, there is no label prefix, therefore the method name and label name are identical.

Parse Query String	No	No	False	This property causes any query string that is present with an incoming message to be parsed and decoded (according to `http://tools.ietf.org/html/rfc3986`) into the following location in the local environment as a series of name-value elements that match the names and values present in the query string:
				
				`LocalEnvironment.HTTP.Input.QueryString`
				
				For example, for this query string:
				
				`?myParam1=my%22Value%221&myParam2=my%22Value%222`
				the following elements are placed into the local environment under the QueryString folder:
				
				`myParam1 with a value of my"Value"1`
				`myParam2 with a value of my"Value"2`

The HTTPInput node Input Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	BLOB	The domain that is used to parse the incoming message. If you leave this field blank, the default value is BLOB. Select the name of the parser that you are using from the list: • MRM • XMLNSC • XMLNS • MIME • BLOB • XML (this domain is deprecated; use XMLNSC) You can also specify a user-defined parser, if appropriate.
Message set	No	No		The name or identifier of the message set in which the incoming message is defined. All available message sets are in the list. If you are using the MRM parser or the XMLNSC parser in validating mode, select the Message set that you want to use. This list is populated with available message sets when you select MRM or XMLNSC as the domain. If you set this property, then later update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Message type	No	No		The name of the incoming message. If you are using the MRM parser, select the type of message from the list in Message type. This list is populated with messages that are defined in the Message set that you have selected.
Message format	No	No		The name of the physical format of the incoming message. If you are using the MRM parser, select the format of the message from the list in Message format. This list includes all the physical formats that you have defined for this Message set.

The properties of the Parser Options for the HTTPInput node are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	On Demand	This property controls when an input message is parsed. Valid values are On Demand, Immediate, and Complete. By default, this property is set to On Demand, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see “Parsing on demand” on page 1449.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the XMLNSC parser creates syntax elements in the message tree with data types taken from the XML Schema. You can select this property only if you set the Validate property on the **Validation** tab to Content or Content and Value.
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the message data appears under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or the input message Parsing property Message domain is XMLNS.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in an input message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.

Property	M	C	Default	Description
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in an input message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in an input message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the input message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if Validate is None); entries that are specified in Opaque Elements are ignored if validation is enabled.

The HTTPInput node Error handling properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Maximum client wait time (sec)	Yes	Yes	180	The length of time, in seconds, for which the TCP/IP listener that received the input message from the Web service client waits for a response from the HTTPReply node in the message flow. The valid range is zero (which means a short wait) through (2^{31})-1. If a response is received within this time, the listener propagates the response to the client. If a response is not received in this time, a fault message is generated indicating that the timeout has expired.	
Fault format	No	Yes	SOAP 1.1	The format of any HTTP errors that are returned to the client. Valid values are SOAP 1.1, SOAP 1.2, and HTML.	faultFormat

The Validation properties of the HTTPInput node are described in the following table. If a message is propagated to the Failure terminal of the node, it is not validated. For more details, see “Validating messages” on page 204 and “Validation properties” on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are None, Content and Value, and Content.	validatemaster
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Security properties of the HTTPInput node are described in the following table. Set values for the properties that control the extraction of an identity from a message when a security profile is associated with the node. For more information

about these properties, see Identity, Configuring the extraction of an identity, Message flow security , and Setting up message flow security

Property	M	C	Default	Description
Identity token type	No	No	None	This property specifies the type of identity token that is present in the incoming message. Valid values are: • Transport Default • Username • Username + Password • X.509 Certificate If this property is not specified, the identity is retrieved from the Basic-Auth transport header and the type is set to Username + Password.
Identity token location	No	No	None	This property specifies where, in the message, the identity can be found. The location is specified as an ESQL field reference or XPath expression. If this property is not specified, the identity is retrieved from the Authorization Transport headers.
Identity password location	No	No	None	This property specifies where, in the message, the password can be found. The location is specified as an ESQL field reference or XPath expression. If you do not specify a value for this property, the password is retrieved from the Authorization Transport headers. You can set this property only if the Identity type is set to Username + Password.
Identity IssuedBy location	No	No	None	This property specifies a string or path expression that describes the issuer of the identity. The location is specified as an ESQL field reference or XPath expression. If you do not specify a value for this property, the default value is the name of the User Agent, or, if this name is not set, the string HTTP.
Treat security exceptions as normal exceptions	No	No	False	This property specifies whether to treat security exceptions (such as Access Denied) as normal exceptions, and propagate them to the Failure terminal (if wired). This option is turned off by default, which ensures that security exceptions cause the message to be backed out even if the Failure terminal is wired.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

HTTPReply node

Use the HTTPReply node to return a response from the message flow to an HTTP client. This node generates the response to an HTTP client from which the input message was received by the HTTPInput node, and waits for confirmation that it has been sent.

This topic contains the following sections:

- "Connecting the output terminals to another node"
- "Terminals and properties"

Purpose

The HTTPReply node can be used in a message flow that sends a response to an inbound HTTP or HTTPS messages. The most common example of this scenario is a message flow that implements a Web service.

For more information about Web services, see "Working with Web services" on page 735.

Because all HTTP messages are handled by a single broker-wide listener, you must deploy the flow that includes the HTTPReply node to the same broker. The execution group is not significant, because the listener is shared.

The HTTPReply node constructs a reply message for the Web service client from the entire input message tree, and returns it to the requester. If the message was initially received by an HTTPInput node in another message flow, the response is associated with the reply by a request identifier that is stored in the local environment of the message by the HTTPInput node.

The HTTPReply node is contained in the **HTTP** drawer of the palette, and is represented in the workbench by the following icon:

Connecting the output terminals to another node

Connect the Out or Failure terminal of this node to another node in this message flow if you want to process the message further, process errors, or send the message to an additional destination.

Terminals and properties

When you have put an instance of the HTTPReply node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The HTTPReply node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if a failure is detected when the message is propagated.
Out	The output terminal to which the message is routed if it has been propagated successfully, and if further processing is required in this message flow.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a

value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file for deployment).

The HTTPReply node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	HTTPReply	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The HTTPReply node Basic properties are described in the following table.

Property	M	C	Default	Description
Ignore transport failures	Yes	No	Selected	Select Ignore transport failures if you want transport-related failures to be ignored (for example, if the client is disconnected). If you clear the check box, and a transport-related error occurs, the input message is propagated to the Failure terminal. If you clear the check box, you must supply a value for Reply send timeout (sec).
Reply send timeout (sec)	Yes	No	120	Set the Reply send timeout (sec) value if you are not ignoring transport failures. This property specifies the length of time, in seconds, that the node waits for an acknowledgment that the client has received the reply. If the acknowledgment is received within this time, the input message is propagated through the Out terminal to the rest of the message flow, if it is connected. If an acknowledgment is not received within this time, the input message is propagated through the Failure terminal, if it is connected. If the Failure terminal is not connected, and an acknowledgment is not received in time, an exception is generated. The valid range is zero (which means an indefinite wait) to (2^{31})-1. This property is valid only if Ignore transport failures is cleared.
Generate default HTTP headers from reply or response	Yes	No	Selected	Select Generate default HTTP headers from reply or response if you want the default Web service headers to be created using values from the HTTPReplyHeader or the HTTPResponseHeader. If the appropriate header is not present in the input message, default values are used. The node always includes, in the HTTPReplyHeader, a Content-Length header, which is set to the correct calculated value, even if this header was not included in the original request.

The Validation properties of the HTTPReply node are described in the following table.

If a message is propagated to the Failure terminal of the node, it is not validated. For a full description of these properties, see "Validation properties" on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	Inherit	This property controls whether validation takes place. Valid values are None, Content and Value, Content, and Inherit.	validateMaster

Property	M	C	Default	Description	mqsiapplybaroverride command property
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

HTTPRequest node

Use the HTTPRequest node to interact with a Web service.

This topic contains the following sections:

Purpose

The HTTPRequest node interacts with a Web service, using all or part of the input message as the request that is sent to that service. You can also configure the node to create a new output message from the contents of the input message, augmented by the contents of the Web service response, before you propagate the message to subsequent nodes in the message flow.

Depending on the configuration, this node constructs an HTTP or an HTTP over SSL (HTTPS) request from the specified contents of the input message, and sends this request to the Web service. The node receives the response from the Web service, and parses the response for inclusion in the output tree. The node generates HTTP headers if these are required by your configuration.

You can use this node in a message flow that does or does not contain an HTTPInput or HTTPReply node.

The HTTPRequest node handles messages in the following message domains:
- MRM
- XMLNSC
- XMLNS
- MIME
- BLOB
- XML (this domain is deprecated; use XMLNSC)

The HTTPRequest node is contained in the **HTTP** drawer of the palette, and is represented in the workbench by the following icon:

Using the HTTPRequest node to issue a request to a Web service

An HTTP request has two parts:

1. The URL of a service.
2. A stream of data that the remote server processes, then sends back a response, which is often a SOAP or other Web service message in XML.

The URL is of the format `http://<address>[:port]/<function>`; for example, `http://localhost:7080/request`. This URL can be specified statically in the HTTPRequest node parameters as a field in the message itself, or as a field in the local environment. The data to be sent to the Web service can be the whole, or a portion of, the message tree, as specified in the HTTPRequest node properties.

The data must be in CCSID 1208 format for most requests. The reply can replace the input message, or be inserted into the message tree; the location is specified in the HTTPRequest node parameters. The domain for the reply is XMLNS. If the request is successful, the HTTPResponse is inserted into the front of the message tree, the reply placed in the specified location in the tree, and the request propagated to the Out terminal. If the HTTPRequest node is not able to issue the request, an ExceptionList is inserted into the message tree and the tree is propagated to the Failure terminal.

If the request is sent successfully by the HTTPRequest node, but the Web service is not successful, the HTTPResponse is inserted into the message tree, and propagated to the Error terminal. The error message location parameter on the HTTPRequest node specifies where in the tree the response is placed, for example `OutputRoot.XMLNS.error`. You might need to use a Compute node to cast this response to an appropriate code page to be able to display the data, for example:

```
Set OutputRoot.XMLNS.error850 = CAST(InputRoot.XMLNS.error.BLOB as CHAR CCSID 850);
```

For information about HTTP, see Hypertext Transfer Protocol - HTTP/1.1. For more information about HTTP return codes, see HTTP Response codes.

You can specify a timeout interval, so that if the request takes longer than the specified duration, the request is propagated to the Error terminal with an appropriate message. For each request that the HTTPRequest node processes, it opens a connection, and then closes it when the response is returned. If the timeout interval is specified, the socket is closed after the interval. This closure ensures that a request gets only the correct response, and any response data for a request that has timed out is discarded.

You can use the HTTP proxy to route a request through an intermediate site. You can run tools as a proxy to see the request and the response, and therefore debug your flows. The HTTP destination is as seen by the proxy; if you specify the HTTP destination of localhost, and the HTTP proxy is running on a different computer, the request is routed to the remote proxy computer, not the computer from which the original request was issued.

Using the HTTPRequest node in a message flow

The HTTPRequest node can be used in any message flow that needs to send an HTTP request. The most common example of this is a message flow that calls a Web service.

For more information about Web services, see “Working with Web services” on page 735.

Handling errors

The node interacts directly with an external service using TCP/IP; it can, therefore, experience the following types of error:

- Errors that are generated by TCP/IP, for example `no route to host` or `connection refused`.

  If the node detects these errors, it generates an exception, populates the exception list with the error information that is received, and routes the input message unchanged to the Failure terminal.
- Errors that are returned by the Web server. These errors are represented by HTTP status codes that are outside the range 100 - 299. If the node detects these errors, it routes the reply to the Error terminal while following the properties specified on the **Error** tab.

  The reply is produced as a BLOB message because the node cannot determine in what format the reply will be. If you have not configured this node to handle redirection, messages with a redirection status code (3xx) are also handled in the same way.

HTTP Response Codes

The HTTPRequest node treats the 100 series status codes as a 'continue' response, discards the current response, and waits for another response from the Web server.

The 200 series status codes are treated as success, the settings on the various tabs on the node determine the format of the output message that is generated, and the response is routed to the Out terminal of the node.

The 300 series status codes are for redirection. If the Follow HTTP(s) Redirection property is selected, the node does not resend the request to the new destination that is specified in the response that is received. If the Follow HTTP(s) Redirection property is not selected, the codes are treated as an error, as described in “Using the HTTPRequest node to issue a request to a Web service” on page 992. For more information about HTTP return codes, see HTTP Response codes.

The 400 and 500 series status codes are errors, and are treated as described in “Using the HTTPRequest node to issue a request to a Web service” on page 992. For more information about HTTP return codes, see HTTP Response codes.

Manipulating headers

If you select Replace input message with web-service response or Replace input with error, the header for the input message (the header that belongs to the message when it arrives at the In terminal of the HTTPRequest node) is not propagated with the message that leaves the HTTPRequest node. However, if one of the properties that specify a location in the message tree is specified, the input message headers are propagated.

The HTTPResponse header, which contains the headers that are returned by the remote Web service, is the first header in the message (after Properties) that is propagated from the node. This action is taken regardless of the options that are chosen. Therefore, for the reply from the HTTPRequest node to be put to a WebSphere MQ queue, manipulate the headers so that an MQMD is the first header (after Properties).

If you are replacing the input message with a response, you can copy the input message MQMD to the Environment tree before the HTTPRequest node, and then copy it back into the message tree after the HTTPRequest node. If you are specifying a location for the response, in order to maintain existing input message headers, you must move or remove the HTTP Response header so that the MQMD is the first header.

The following example contains ESQL that removes the HTTPHeader:

```
SET OutputRoot = InputRoot;
SET OutputRoot.HTTPResponseHeader = NULL;
```

The following example contains ESQL for moving the HTTPHeader, and therefore preserving the information that it provides:

```
SET OutputRoot = InputRoot;
DECLARE HTTPHeaderRef REFERENCE TO OutputRoot.HTTPResponseHeader;
DETACH HTTPHeaderRef;
ATTACH HTTPHeaderRef TO OutputRoot.MQMD AS NEXTSIBLING;
```

Configuring the HTTPRequest node

When you have put an instance of the HTTPRequest node into a message flow, you can configure the node; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view.

All mandatory properties, for which you must enter a value, are marked with an asterisk.

Configure the HTTPRequest node:

1. Optional: On the **Description** tab, enter a Short description, a Long description, or both. You can also rename the node on this tab.
2. On the **Basic** tab:
   a. The HTTPRequest node determines the URL for the Web service to which it sends a request. Set one of the following three options; the node checks these in the order shown (that is, the first always overrides the second, the second overrides the third):
      1) X-Original-HTTP-URL in the HTTPRequest header in the input message
      2) LocalEnvironment.Destination.HTTP.RequestURL in the input message
      3) The Web service URL property

      The first two options provide dynamic methods to set a URL for each input message as it passes through the message flow. To use either of these options, include a Compute node in the message flow, before the HTTPRequest node, to create and initialize the required value.

      The third option provides a value that is fixed for every message that is received in this node. Set this property to contain a default setting that is used if the other fields have not been created, or contain a null value. If either field contains a value, the setting of this property is ignored. The Web service URL property must contain a valid URL or the deployment fails. Ensure that the value that you set in X-Original-HTTP-URL or the

LocalEnvironment.Destination.HTTP.RequestURL is also a valid URL; if it is not, the node uses the default setting from the Web service URL property.

If a URL begins `http://`, the request node makes an HTTP request to the specified URL. If the URL begins `https://`, the request node makes an HTTP over SSL (HTTPS) request to the specified URL, using the parameters that are specified on the **SSL** tab for the node.

b. Set the value of the Request timeout (sec) property, which is the length of time, in seconds, that the node waits for a response from the Web service. If a response is received within this time, the reply is propagated through the Out terminal to the rest of the message flow. If a response is not received within this time, the input message is propagated through the Failure terminal, if it is connected. If the Failure terminal is not connected, and a response is not received in this time, an exception is generated.

3. On the **HTTP Settings** tab:
   a. In HTTP(S) proxy location, set the location of the proxy server to which requests are sent.
   b. Select Follow HTTP(S) redirection to specify how the node handles Web service responses that contain an HTTP status code of 300 to 399:
      - If you select the check box, the node follows the redirection that is provided in the response, and reissues the Web service request to the new URL (included in the message content).
      - If you clear the check box, the node does not follow the redirection provided. The response message is propagated to the Error terminal.
   c. Select one of the options for the HTTP version property. Valid values are: 1.0 or 1.1.

      If you select the HTTP version property value 1.1, you can also select Enable HTTP/1.1 keep-alive.
   d. Select one of the options for the HTTP method property. Valid values are: POST, GET, PUT, DELETE, and HEAD.
4. On the **SSL** tab, if you want to use HTTP over SSL (HTTPS) requests, set the values for HTTPS requests:
   a. Specify the Protocol property that you want to use to make the request. Both ends of an SSL connection must agree on the protocol to use. Therefore, the chosen protocol must be one that the remote server can accept. The following options are available:
      - SSL. This option is the default. This option tries to connect using the SSLv3 protocol first, but allows the handshake to fall back to the SSLv2 protocol where the SSLv2 protocol is supported by the underlying JSSE provider.
      - SSLv3. This option tries to connect with the SSLv3 protocol only. Fallback to SSLv2 is not allowed.
      - TLS. This option tries to connect with the TLS protocol only. Fallback to SSLv3 or SSLv2 is not allowed.
   b. Set the Allowed SSL ciphers property. Use this setting to specify a single cipher (such as SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA) or a list of ciphers that are the only ones used by the connection. This set of ciphers must include one or more that are accepted by the remote server. A comma is used as a separator between the ciphers. The default value is an empty string, which enables the node to use any, or all, of the available ciphers during the SSL connection handshake. This method gives the greatest scope for making a successful SSL connection.

5. On the **Response Message Parsing** tab, set values for the properties that describe the message domain, message set, message type, and message format that the node uses to determine how to parse the response message returned by the Web service. If an error message is returned by the Web service, the values of these properties are ignored, and the message is parsed by the BLOB parser.
   a. In Message domain, select the name of the parser that you are using from the list. If the field is blank, the default value is BLOB. Choose from the following options:
      - MRM
      - XMLNSC
      - XMLNS
      - MIME
      - BLOB
      - XML (this domain is deprecated; use XMLNSC)

      You can also specify a user-defined parser, if appropriate.
   b. If you are using the MRM parser or the XMLNSC parser in validating mode, select the relevant Message set from the list. This list is populated with available message sets when you select MRM or XMLNSC as the Message domain.
   c. If you are using the MRM parser, select the correct message from the list in Message type. This list is populated with messages that are defined in the Message set that you have selected.
   d. If you are using the MRM parser, select the format of the message from the list in Message format. This list includes all the physical formats that you have defined for this Message set.
6. On the **Parser Options** subtab:
   a. Parse timing is, by default, set to On Demand, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see "Parsing on demand" on page 1449.
   b. If you are using the XMLNSC parser, set values for the properties that determine how the XMLNSC parser operates. For more information, see "Manipulating messages in the XMLNSC domain" on page 408.
7. On the **Error Handling** tab, set values for the properties that determine how an error message returned by the Web service is handled:
   a. For the whole Web service error message to be propagated as the output message, leave Replace input with error selected (the default setting).

      For the Web service error message to be included in the output message with part of the input message content, clear Replace input with error and set the Error message location property. If you clear this property, the node copies the input message to the output message and writes the Web service error message over the output message content at the specified location (the input message itself is not modified).
   b. In the Error message location field, enter the start location (within the output message tree) at which the parsed elements from the Web service error message bit stream are stored. This property is required only if you have cleared Replace input with error.

      You can enter any valid ESQL field reference, including expressions within the reference and new field references (to create a new node in the message tree for the response). For example, enter:

      ```
      OutputRoot.XMLNSC.ABC.DEF
      ```

      or

```
Environment.WSError
```

If you select Replace input with error, this property is ignored.

8. On the **Advanced** tab, set values for the Advanced properties that describe the structure and content of the Web service request and response:
   a. Specify the content of the request message that is sent to the Web service:
      - For the request message to be the whole input message body, leave Use whole input message as request selected (the default setting).

        For the request message to contain a subset of the input message, clear Use whole input message as request and set the Request message location in tree property.
      - In the Request message location in tree field, enter the start location from which the content of the input message tree is copied to the request message. This property is required only if you have cleared Use whole input message as request. The node creates a new request message and copies the specified parts of the input message (the input message itself is not modified).

        You can enter any valid ESQL field reference, including expressions within the reference. For example, enter:

        ```
        InputRoot.XMLNSC.ABC
        ```

        If you select Use whole input message as request, this property is ignored.

      When the appropriate message tree content is parsed to create a bit stream, the message properties (Message domain, Message set, Message type, and Message format) that are associated with the input message body and stored in the Properties folder are used.
   b. Specify the content of the output message that is propagated to the next node in the message flow:
      - For the whole Web service response message to be propagated as the output message, leave Replace input message with web-service response selected (the default setting).

        For the Web service response message to be included in the output message with part of the input message content, clear Replace input message with web-service response and set the Response message location in tree property. If you clear this property, the node copies the input message to the output message and writes the Web service response message over the output message content at the specified location (the input message itself is not modified).
      - In the Response message location in tree field, enter the start location (within the output message tree) at which the parsed elements from the Web service response message bit stream are stored. This property is required only if you have cleared Replace input message with web-service response.

        You can enter any valid ESQL field reference, including expressions within the reference, and including new field references (to create a new node in the message tree for the response). For example, enter:

        ```
        OutputRoot.XMLNSC.ABC.DEF
        ```

        or

        ```
        Environment.WSReply
        ```

        If you select Replace input message with web-service response, this property is ignored.

When the response bit stream is parsed to create message tree contents, the message properties (Message domain, Message set, Message type, and Message format), that you have specified in the Response Message Parsing properties of the node, are used.

c. For the node to generate an HTTPRequestHeader for the request message, leave Generate default HTTP headers from input selected (the default setting).

   If you do not want the node to generate an HTTPRequestHeader for the request message, clear Generate default HTTP headers from input. To control the contents of the HTTPRequestHeader that is included in the request message, include a Compute node that adds an HTTPRequestHeader to the input message before this HTTPRequest node in the message flow, and clear this check box.

   - If you have selected Generate default HTTP headers from input and the input message includes an HTTPRequestHeader, the HTTPRequest node extracts Web service headers from the input HTTPRequestHeader and adds any unique Web service headers, except Host (see the following table), that are present in an HTTPInputHeader, if one exists in the input message. (An HTTPInputHeader might be present if the input message has been received from a Web service by the HTTPInput node.)

     The HTTPRequest node also adds the Web service headers shown in the following table, with default values, if these are not present in the HTTPRequestHeader or the HTTPInputHeader.

Header	**Default value**
SOAPAction	"" (empty string)
Content-Type	text/xml; charset=*ccsid of the message body*
Host	The host name to which the request is to be sent.

     The HTTPRequest node also adds the optional header Content-Length with the correct calculated value, even if this value is not present in the HTTPRequestHeader or the HTTPInputHeader.

   - If you have selected Generate default HTTP headers from input and the input message does not include an HTTPRequestHeader, the HTTPRequest node extracts Web service headers, except Host, from the HTTPInputHeader (if it is present in the input message). The HTTPRequest node adds the required Web service headers with default values, if these values are not present in the HTTPInputHeader.
   - If you have cleared Generate default HTTP headers from input and the input message includes an HTTPRequestHeader, the node extracts all Web service headers present in the input HTTPRequestHeader. The node does not check for the presence of an HTTPInputHeader in the input message, and it does not add the required Web service headers if they are not supplied by the input HTTPRequestHeader.
   - If you have cleared Generate default HTTP headers from input and the input message does not include an HTTPRequestHeader, no Web service headers are generated. The HTTPRequest node does not check for the presence of an HTTPInputHeader in the input message and does not add any required Web service header. The request message is propagated to the Web service without an HTTPRequestHeader. This action typically causes an error to be generated by the Web service, unless the Web service is configured to handle the message contents.

9. On the **Validation** tab, set Validation properties if you want the parser to validate the body of response messages against the Message set. (If a message is propagated to the Failure terminal of the node, it is not validated.) These properties do not cause the input message to be validated. It is expected that, if such validation is required, the validation has already been performed by the input node or a preceding validation node.

   For more details see "Validating messages" on page 204 and "Validation properties" on page 1445.

Connecting the output terminals to another node

Connect the Out, Error, or Failure terminal of this node to another node in this message flow to process the message further, to process errors, or to send the message to an additional destination. If you do not connect the Error terminal, the message is discarded. If you do not connect the Failure terminal, the broker provides default error processing, see "Handling errors in message flows" on page 244.

Terminals and properties

The HTTPRequest node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if a failure is detected during processing in the node.
Out	The output terminal to which the message is routed if it represents successful completion of the Web service request, and if further processing is required within this message flow.
Error	The output terminal to which messages that include an HTTP status code that is not in the range 200 through 299, including redirection codes (3xx) if you have not set the property Follow HTTP(s) redirection property, is routed.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk on the panel if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the broker archive file to deploy it).

The HTTPRequest node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, HTTPRequest	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The HTTPRequest node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Web service URL	Yes	Yes		The URL for the Web service. You must provide this in the form `http://hostname[:port]/[path]` where • `http://hostname` must be specified. • *port* has a default of 80. If you specify a value, you must include the : before the port number. • *path* has a default of /. If you specify a value, you must include the / before the path.	URLSpecifier
Request timeout (sec)	Yes	Yes	120	The time in seconds that the node waits for a response from the Web service. The valid range is 1 through (2^{31})-1. You cannot enter a value that represents an unlimited wait. The timeout might take up to one second longer than the specified value.	timeoutForServer

The HTTPRequest node HTTP Settings properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
HTTP(S) proxy location	No	Yes		The proxy server to which requests are sent. This value must be in the form hostname:port.	httpProxyLocation
Follow HTTP(S) redirection	No	No	Cleared	If you select the check box, redirections are followed. If you clear this check box, redirections are not followed.	
HTTP version	No	Yes	1.0	The HTTP version to use for requests. Valid values are 1.0 and 1.1.	httpVersion
Enable HTTP/1.1 keep-alive	No	Yes	Selected (if HTTP version is 1.1)	Use HTTP/1.1 Keep-Alive.	enableKeepAlive
HTTP method	No	No	POST	The HTTP method. Valid values are POST, GET, PUT, DELETE, and HEAD. By default, the HTTPRequest node uses the HTTP POST method when it connects to the remote Web server. HEAD is used to determine whether a service is available - for example, by a Network Dispatcher trying to work out which servers are available - and sends back the correct headers (including content-length) but no body data.	

The HTTPRequest node SSL properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Protocol	No	Yes	SSL	The SSL protocol to use when making an HTTPS request.	protocol
Allowed SSL ciphers	No	Yes		A comma-separated list of ciphers to use when making an SSL request. The default value of an empty string means use all available ciphers.	allowedCiphers

The HTTPRequest node Response Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	BLOB	The domain that is used to parse the response message that is received from the Web service. If the field is blank then the default is BLOB.
Message set	No	No		The name or identifier of the message set in which the response message is defined. If you set this property, and then update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Message type	No	No		The name of the response message.
Message format	No	No		The name of the physical format of the response message.

The HTTPRequest node Parser Options properties are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	On Demand	This property controls when a response message is parsed. Valid values are On Demand, Immediate, and Complete. For a full description of this property, see "Parsing on demand" on page 1449.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the XMLNSC parser creates syntax elements in the message tree with data types taken from the XML Schema. You can select this property only if you set the Validate property on the **Validation** tab to Content or Content and Value.
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the response message data appears under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or Response Message Parsing properties Domain is XMLNS.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in a response message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in a response message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in a response message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.

Property	M	C	Default	Description
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the response message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if the Validate property is set to None); entries that are specified in Opaque Elements are ignored if validation is enabled.

The HTTPRequest node Error Handling properties are described in the following table.

Property	M	C	Default	Description
Replace input with error	No	No	Selected	If you select this check box, the input message content is replaced by the error message content. If you clear this check box, you must specify Error message location.
Error message location	Yes	No	OutputRoot	The start location at which the parsed elements from the Web service error bit stream are stored. This property takes the form of an ESQL field reference.

The HTTPRequest node Advanced properties are described in the following table.

Property	M	C	Default	Description
Use whole input message as request	No	No	Selected	If you select this check box, the whole input message body is to be passed to the Web service. If you clear this check box, you must select Request message location in tree.
Request message location in tree	Yes	No	InputRoot	The start location from which the bit stream is created for sending to the Web service. This property takes the form of an ESQL field reference.
Replace input message with web-service response	No	No	Selected	If you select this check box, the Web service response message replaces the copy of the input message as the content of the output message that is created. If you clear this check box, you must select Response message location in tree.
Response message location in tree	Yes	No	OutputRoot	The start location at which the parsed elements from the Web service response bit stream are stored. This property takes the form of an ESQL field reference.
Generate default HTTP headers from input	No	No	Selected	If you select this check box, an HTTPRequestHeader is generated. If you clear this check box, a valid HTTPRequestHeader must exist in the input message.

The HTTPRequest node Validation properties are described in the following table.

For a full description of these properties see “Validation properties” on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are None, Content and Value, Content, and Inherit.	validateMaster

Property	M	C	Default	Description	mqsiapplybaroverride command property
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Local environment overrides

You can dynamically override set values in the local environment in the same way as setting values in other elements of a message. The following values can be set under LocalEnvironment.Destination.HTTP.

Setting	Description
RequestURL	Overrides the Web service URL property on the node. For example: `SET OutputLocalEnvironment.Destination.HTTP.RequestURL = 'http://ibm.com/abc/';`
Timeout	Overrides the Request timeout (sec) property on the node. For example: `SET OutputLocalEnvironment.Destination.HTTP.Timeout = 42;`
ProxyURL	Overrides the HTTP(S) proxy location property on the node. For example: `SET OutputLocalEnvironment.Destination.HTTP.ProxyURL = 'my.proxy';`
RequestLine.RequestURI	Overrides the `RequestURI`, which is the path after the URL and port. For example: `SET OutputLocalEnvironment.Destination.HTTP.RequestLine.RequestURI = '/abc/def';`
RequestLine.HTTPVersion	Overrides the HTTP version property on the node. For example: `SET OutputLocalEnvironment.Destination.HTTP.RequestLine.HTTPVersion = 'HTTP/1.1';`
KeepAlive	Overrides the Enable HTTP/1.1 keep-alive property on the node. For example: `SET OutputLocalEnvironment.Destination.HTTP.KeepAlive = TRUE;`
RequestLine.Method	Overrides the HTTP method property on the node. For example: `SET OutputLocalEnvironment.Destination.HTTP.RequestLine.Method = 'GET';`
SSLProtocol	Overrides the SSLProtocol. For example: `SET OutputLocalEnvironment.Destination.HTTP.SSLProtocol = 'TLS';` Valid values are: SSL, SSLv3, and TLS.
SSLCiphers	Overrides the Allowed SSL Ciphers property on the node. For example: `SET OutputLocalEnvironment.Destination.HTTP.SSLCiphers = 'SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA';`

Setting	Description

ProxyConnectHeaders	Specifies additional headers that are used if the outbound request is an SSL connection through a proxy. These additional headers are sent with the initial CONNECT request to the proxy. For example, you can send proxy authentication information to a proxy server when you are using SSL. You can send multiple headers but each one must be separated by a carriage return and a line feed (ASCII 0x0D 0x0A), in accordance with RFC2616; for example:
	`DECLARE CRLF CHAR CAST(X'0D0A' AS CHAR CCSID 1208);`
	`SET OutputLocalEnvironment.Destination.HTTP.ProxyConnectHeaders =`
	`'Proxy-Authorization: Basic Zm5lcmJsZTpwYXNzd29yZA==' || CRLF ||`
	`'Proxy-Connection: Keep-Alive' || CRLF;`
	
	This setting is used only if the request is an SSL request through a proxy server. To send proxy authentication information for a non-SSL request, specify the individual headers in the HTTPRequestHeader folder, as shown in the following example:
	`SET OutputRoot.HTTPRequestHeader."Proxy-Authorization" =`
	`'Basic Zm5lcmJsZTpwYXNzd29yZA==';`
	`SET OutputRoot.HTTPRequestHeader."Proxy-Connection" = 'Keep-Alive';`

UseFolderMode	Sets the `UseFolderMode`. Use for bitstream generation; for certain parsers this changes the output bitstream. For example: `SET OutputLocalEnvironment.Destination.HTTP.UseFolderMode = TRUE;`

QueryString	Allows the setting of outbound query string parameters. Each parameter must be be set individually. For example:
	`SET OutputLocalEnvironment.Destination.HTTP.QueryString.param1 = 'my"Value"1';`
	`SET OutputLocalEnvironment.Destination.HTTP.QueryString.param2 = 'my"Value"2';`
	
	The above ESQL results in the following query string being encoded (according to `http://tools.ietf.org/html/rfc3986`) and sent with the outbound request:
	`?param1=my%22Value%221&param2= my%22Value%222`
	
	If the destination URL already has one or more query parameters, additional parameters specified here are appended to the end of the existing list.

Working with WrittenDestination data

After the request has been made, the WrittenDestination folder in the local environment is updated with the URI to which the request was sent. A WrittenDestination for an HTTPRequest node has the following format:

```
WrittenDestination = (
   HTTP  = (
      RequestURL = 'http://server:port/folder/page'
   )
)
```

IMSRequest node

Use the IMSRequest node to send a request to run a transaction on a local or remote IMS system, and wait for a response. IMS Connect must be configured and running on the IMS system.

This topic contains the following sections:

Purpose

The following example illustrates a situation in which you would use an IMSRequest node.

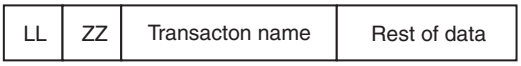
WebSphere Message Broker can be used to expose an existing target IMS banking application as a Web Service. For example, the IMS application provides transactions that operate on a database that contains information about customers' bank accounts. In this example, the Web service consumer sends a SOAP message across HTTP to WebSphere Message Broker and synchronously waits for the response. The WebSphere Message Broker message flow transforms the SOAP message to IMS format (including the LLZZ and transaction code fields), then sends that bit stream to IMS. The message flow waits for a response. IMS schedules the destination program and queues the request data for that program. The target program accesses the customer account database, builds a response message that consists of the account statement, and returns it to the WebSphere Message Broker message flow. The message flow transforms the IMS format to a SOAP format and sends that SOAP response back across HTTP to the Web service consumer.

The IMSRequest node is contained in the **IMS** drawer of the message flow node palette, and is represented in the workbench by the following icon:

To enable function that becomes available in WebSphere Message Broker fix packs, use the **-f** parameter on the mqsichangebroker command. For more information, see mqsichangebroker command.

Using the IMSRequest node in a message flow

The IMSRequest node sends requests to IMS by using IMS Connect. The node takes the incoming message's bit stream and sends it to IMS. It then receives a bit stream, which it passes to the response parser. The bit stream that is sent to IMS must conform to the format that is shown in the following diagram:



The message flow that contains the IMSRequest node ensures that the message that is received by the IMSRequest node has this structure where:

- *LLZZ* is a four-byte field. The first two bytes indicate the length of the bit stream, and the other two bytes are reserved for use by IMS.
- For request segments, the transaction code must follow. The transaction code can contain up to eight characters; if it contains less than eight characters, the transaction code must be delimited by a space. The response segments do not need to have the transaction name, but an IMS program can add it.
- The rest of the data comprises anything else that the IMS program needs.

The IMS program produces many messages. You can receive all messages as a single transmission bit stream, or you can receive them separately. Each message can contain multiple segments; all segments for each message are returned at the same time.

The IMSRequest node has two modes of operation, which you specify by selecting or clearing the Use connection properties defined on node check box. If you select the check box, all properties are taken from the node by using the following properties in the node connection details section:
- Hostname
- Port number
- Data store name

If you clear the Use connection properties defined on node check box, all connection details are retrieved from the configurable services. However, if the Security identity property is set, the security identity on the configurable service is ignored and the value of the node property is used instead.

View the following sample to see how to use this node:
- IMS Synchronous Request

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

Using configurable services for IMS nodes

You can configure IMS nodes to get connection details from a configurable service. For details about creating, changing, reporting, and deleting the configurable services, see "Changing connection information for the IMSRequest node" on page 223.

Terminals and properties

When you have put an instance of the IMSRequest node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The IMSRequest node terminals are described in the following table.

Terminal	Description
In	The input terminal that receives the message that triggers the node.
Out	The output terminal to which the node sends a message after it has been received from the external resource. The message is sent to the terminal unchanged, except for some added status information.
Failure	If an error occurs in the IMSRequest node, the message is sent to the Failure terminal.
Timeout	The output terminal to which the message is sent if a timeout occurs. The input message is propagated to this terminal with an exception list that describes the timeout. If the Timeout terminal is not connected and a timeout occurs, the message is routed to the Failure terminal. A timeout can occur in either of the following situations: • The IMS program has not responded by the time the execution timeout has expired. The execution timeout is configured by using the Timeout waiting for a transaction to be executed property on the IMSRequest node. • WebSphere Message Broker has not received the response across the TCP/IP network by the time the socket timeout expires. You can configure the socket timeout on the configurable service.

The following tables describe the node properties. The columns headed M indicate whether the property is *mandatory* (marked with an asterisk on the panel if you must enter a value when no default is defined); the columns headed C indicate

whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The IMSRequest node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, IMSRequest	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The IMSRequest node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Use connection properties defined on node	No	Yes	Selected	If you select this check box, the connection properties that are defined on the node are used instead of a configurable service and security identity that are defined on the broker. If you clear this check box, you must set the Configurable service and Security identity properties.	
Hostname	Yes	Yes		The IP address or host name of the computer that is running the target IMS Connect system. This property is mandatory if the Use connection properties defined on node check box is selected and can be set only if the Use connection properties defined on node check box is selected.	hostname
Port number	Yes	Yes	0	The port number on which IMS Connect is listening for TCP/IP connections. You can obtain the port number from the IMS Connect job log on the IMS system. This property is mandatory only if the Use connection properties defined on node check box is selected and can be set only if the Use connection properties defined on node check box is selected.	portNumber
Data store name	Yes	Yes		The name of the data store that IMS Connect is using. This value must match the ID parameter of the Datastore statement that is specified in the IMS Connect configuration member. This name also serves as the XCF member name for IMS during internal XCF communications between IMS Connect and IMS OTMA. You can obtain the data store name from the IMS Connect job log on the IMS system. This property is mandatory only if the Use connection properties defined on node check box is selected and can be set only if the Use connection properties defined on node check box is selected.	dataStoreName

Property	M	C	Default	Description	mqsiapplybaroverride command property
Timeout waiting for a transaction to be executed	No	No	60	The time (in seconds) that the node waits for IMS to process a transaction. If IMS fails to process a transaction in this time, the node issues an exception, but the connection is not closed. You can set this property only if the Use connection properties defined on node check box is selected. If the check box is cleared, the ExecutionTimeoutSec property on the configurable service is used.	
Configurable service	Yes	Yes		The configurable service from which to get the connection details. All connection details are obtained from the configurable service, except for security information, which is obtained from the Security identity property. This property is mandatory if the Use connection properties defined on node check box is cleared, and can be set only if the Use connection properties defined on node is cleared.	configurableService
Security identity	No	Yes	An empty string	The security identity to look up in the broker to get the user name and password to use. You use the mqsisetdbparms command to set the security identity on the broker. The default value for this property is an empty string, which signifies that the user ID and password are not passed to IMS Connect.	securityIdentity

The IMSRequest node Advanced properties are described in the following table.

Property	M	C	Default	Description
Commit mode	Yes	No	1: SEND_THEN_COMMIT	The commit mode to use when processing IMS transactions. Available values are: • 1: SEND_THEN_COMMIT: (the default value) The transaction is processed using commit mode 1 in IMS. The request transaction is run and data is sent back to the node. After the node acknowledges that it has received the response, the transaction is committed. You cannot process the response before it is committed. The node sends the response after the acknowledgment is sent to IMS. • 0: COMMIT_THEN_SEND: The transaction is processed using commit mode 0 in IMS. The request transaction is run and committed before data is sent back to the node. The node waits for all response messages to be returned before it continues processing.

Property	M	C	Default	Description
Sync level	Yes	No	1: CONFIRM	The synchronization level to use when processing IMS transactions. If Commit mode is set to 0, the Sync level is automatically set to 1: CONFIRM. If Commit mode is set to 1, the Sync level property can have either of the following values: • 1: CONFIRM: (the default value) The node sends an acknowledgment to IMS after it receives the replies. The node then sends the response after the acknowledgment is sent to IMS. If you set the Sync level to 1: CONFIRM, the IMS program is blocked until the IMSRequest node acknowledges the transaction output, which might affect performance. • 0: NONE: The node does not send any acknowledgments. This value is applicable only when Commit mode is set to 1. Typically, Sync level can be set to 0: NONE for read-only types of interactions, such as queries, which do not need an acknowledgment. However, for critical transactions that involve updates and deletions, it is important to be able to acknowledge the output from IMS. If the acknowledgment is not received (for example, because of a connection failure between WebSphere Message Broker and IMS Connect), the transaction is backed out, avoiding the need for a compensation transaction.

The IMSRequest node Request properties are described in the following table.

Property	M	C	Default	Description
Data Location	Yes	No	$Body	The location in the incoming message tree from which data is retrieved to form the request that is sent from the IMSRequest node to IMS. The default value, $Body, represents the incoming message body. You can enter any XPath or ESQL expression that defines the location of the message tree to serialize and send to IMS.

The IMSRequest node Result properties are described in the following table.

Property	M	C	Default	Description
Output data location	No	No	$OutputRoot	The message tree location to which the IMSRequest node copies the response message tree in the outgoing message assembly. The default value, $OutputRoot, replaces the incoming message with the response.
Copy local environment	No	No	Selected	This property controls whether to copy the incoming local environment or propagate the incoming local environment. By default, this check box is selected, which signifies that the local environment is copied so that the incoming local environment is preserved. The additions to the local environment are visible only to nodes downstream of this node. If this check box is cleared, the incoming local environment is used for the outgoing message. Any modifications that are made to the local environment by this node are visible to both downstream and upstream nodes after this node has completed.

The IMSRequest node Response Message Parsing properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Message domain	No	No		The domain to use to parse the message from the external resource's supplied bit stream.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Message set	No	No	Set automatically	The name of the message set in which the incoming message is defined. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.	
Message type	No	No		The name of the response message.	
Message format	No	No		The name of the physical format of the response message.	
Message coded character set ID	Yes	No	EBCDIC (500)	The ID of the coded character set that is used to interpret bytes of the data that is being read. Valid values are EBCDIC (500) and Broker System Default.	messageCodedCharSetIdProperty
Message encoding	Yes	No	Big Endian, with S390 Floating Point (785)	The encoding scheme for numbers and large characters that is used to interpret bytes of the data that is being read. Valid values are: • Little Endian, with IEEE Floating Point (546) • Big Endian, with IEEE Floating Point (273) • Big Endian, with S390 Floating Point (785) • Broker System Determined For more information about encoding, see "Converting data with message flows" on page 165.	messageEncodingProperty

The IMSRequest node Parser Options properties are described in the following table.

Property	M	C	Default	Description
Parse timing	Yes	No	On Demand	This property controls when a response message is parsed. Valid values are On Demand, Immediate, and Complete. For a full description of this property, see "Parsing on demand" on page 1449.
Build tree using XML schema data types	Yes	No	Cleared	This property controls whether the XMLNSC parser creates syntax elements in the message tree with data types taken from the XML Schema. You can select this property only if you set the Validate property on the **Validation** tab to Content or Content and Value.
Use XMLNSC compact parser for XMLNS domain	Yes	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the response message data appears under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or Response Message Parsing properties Domain is XMLNS.

Property	M	C	Default	Description
Retain mixed content	Yes	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in a response message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	Yes	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in a response message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	Yes	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in a response message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the response message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if Validate is None); entries that are specified in Opaque Elements are ignored if validation is enabled.

The IMSRequest node Validation properties are described in the following table.

For a full description of these properties see “Validation properties” on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	Yes	Yes	None	This property controls whether validation takes place. Valid values are None, Content and Value, Content, and Inherit.	validateMaster
Failure action	Yes	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see “Configuring monitoring event sources using monitoring properties” on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Input node

Use the Input node as an In terminal for an embedded message flow (a subflow).

This topic contains the following sections:

Purpose

You can use a subflow for a common task that can be represented by a sequence of message flow nodes. For example, you can create a subflow to increment or decrement a loop counter, or to provide error processing that is common to a number of message flows.

You must use an Input node to provide the In terminal to a subflow; you cannot use a standard input node (a built-in input node such as MQInput, or a user-defined input node).

When you have started your subflow with an Input node, you can connect it to any In terminal on any message flow node, including an Output node.

You can include one or more Input nodes in a subflow. Each Input node that you include provides a terminal through which to introduce messages to the subflow. If you include more than one Input node, you cannot predict the order in which the messages are processed through the subflow.

The Input node is contained in the **Construction** drawer of the palette, and is represented in the workbench by the following icon:

When you select and include a subflow in a message flow, it is represented by the following icon:

When you include the subflow in a message flow, this icon shows a terminal for each Input node that you include in the subflow, and the name of the terminal (which you can see when you hold the mouse pointer over it) matches the name of that instance of the Input node. Give your Input nodes meaningful names that you can recognize easily when you use their corresponding terminal on the subflow node in your message flow.

Using this node in a message flow

Look at the following sample to see how to use this node:
- Error Handler

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

Terminals and properties

When you have put an instance of the Input node into a message flow, you can configure it; see “Configuring a message flow node” on page 276. The properties of the node are displayed in the Properties view.

The Input node terminals are described in the following table.

Terminal	Description
Out	The input terminal that delivers a message to the subflow.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Input node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, Input.	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see “Configuring monitoring event sources using monitoring properties” on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

JavaCompute node

Use the JavaCompute node to work with messages using the Java language.

This topic contains the following sections:

Purpose

Using this node, you can complete the following tasks:

- Use Java to examine an incoming message and, depending on its content, propagate it unchanged to one of the node's two output terminals; the node behaves in a similar way to a Filter node, but uses Java instead of ESQL to determine which output terminal to use.
- Use Java to change part of an incoming message and propagate the changed message to one of the output terminals.
- Use Java to create and build a new output message that is totally independent of the input message.

The Java code that is used by the node is stored in an Eclipse Java project.

The JavaCompute node is contained in the **Transformation** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

The JavaCompute node uses the same execution model as Java user-defined nodes and therefore the restrictions and assumptions associated with Java user-defined nodes also apply to Java code developed for JavaCompute nodes; see Creating a message processing or output node in Java. Only one instance of the JavaCompute node is created regardless of the number of threads running against the flow (either as a result of additional instances or multiple input nodes). Therefore all of your user Java code must be threadsafe and reentrant. For more information see User-defined extensions execution model and Threading considerations for user-defined extensions.

Double-click the JavaCompute node to open the New JavaCompute Node Class wizard. The wizard guides you through the creation of a new Java project and a Java class that contains some skeleton code. This skeleton code is displayed in a Java editor. For more information about creating Java code for a JavaCompute node, and for examples of the skeleton code or template that are provided, see “Creating Java code for a JavaCompute node” on page 521. If it is not the first time that you have double-clicked the node, the Java code is displayed.

The MbJavaComputeNode class contains two methods that you can override: `onInitialize()` and `onDelete()`. If you want the node to perform any cleanup operations, for example closing sockets, include an implementation of the onDelete method:

```
public void onDelete()
{
  // perform node cleanup if necessary
}
```

This method is called by the broker immediately before it deletes the node. The `onInitialize()` method is called by the broker when the node is constructed, after all of the node attributes have been set.

Look at the following sample to see how to use this node.

- JavaCompute Node

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

Specifying Java

Code Java statements to customize the behavior of the JavaCompute node. For example, you can customize the node to create a new output message or messages, using input message or database content (unchanged or modified), or new data. For example, you might want to modify a value in the input message by adding a value from a database, and store the result in a field in the output message.

Code the Java statements that you want in a Java file that is associated with the JavaCompute node.

If a Java file does not already exist for this node, right-click the JavaCompute node and click **Open Java** to create and open a new Java file in the Editor view. If the file exists already, click **Browse** beside the Java Class property to display the JavaCompute Node Type Selection window. When you type at least one character in the **Select** field, matching Java classes are listed. You can use the asterisk (*) to represent any character as part of a search string; for example, *a*b*. Select the appropriate Java class and click **OK**.

Restriction: Do not try to create another instance of a JavaCompute node from Java code; this is not supported.

Terminals and properties

When you have put an instance of the JavaCompute node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. To associate an instance of a JavaCompute node with a Java class, configure the node's properties. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The JavaCompute node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the input message is routed if a failure is detected during the computation. (Even if the Validate property is set, messages that are propagated to the Failure terminal of the node are not validated.)
Out	The output terminal to which the transformed message is routed.
Alternate	An alternative output terminal to which the transformed message can be routed, instead of to the Out terminal.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the JavaCompute node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type: JavaCompute	The name of the node.

Property	M	C	Default	Description
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The JavaCompute node has the Basic property that is described in the following table.

Property	M	C	Default	Description
Java class	Yes	No	None	The name of the Java class that is used in this node. This name must be displayed in the list of Java classes that are available in the project references for the message flow project. To select a file that already exists, click **Browse**. When you type at least one character in the **Select** field, matching Java classes are listed. You can use the asterisk (*) to represent any character as part of a search string; for example, *a*b*. Select the appropriate Java class and click **OK**.

The Parser Options properties for the JavaCompute node are described in the following table.

Property	M	C	Default	Description
Use XMLNSC Compact Parser for XMLNS Domain	No	No	Cleared	Setting this property causes the outgoing MQRFH2 to specify the XMLNS instead of XMLNSC parser, allowing an external application to remain unchanged. If outgoing messages do not contain MQRFH2 headers, this property has no effect.

The Validation properties of the JavaCompute node are described in the following table.

Set the validation properties to define how the message that is produced by the JavaCompute node is validated. These properties do not cause the input message to be validated. It is expected that, if such validation is required, the validation has already been performed by the input node or a preceding validation node. For more details, see “Validating messages” on page 204 and “Validation properties” on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place, and what part of the message is validated. Valid values are None, Content and Value, Content, and Inherit.	validateMaster
Failure action	No	No	Exception	This property controls what happens if a validation failure occurs. You can set this property only if Validate is set to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

JMSHeader node

Use the JMSHeader node to modify contents of the JMS Header_Values and Application properties so that you can control the node's output without programming.

This topic contains the following sections:

Purpose

Use the node to control the output from a JMSOutput node. A subset of common values can be changed in the JMS Header, and user-chosen properties can be added, changed, or deleted for the Application properties.

For JMS Header_Values properties, the node provides a set of fields that you can modify using predefined values, user-defined values, or XPath expressions. XPath is used to provide a valid location from which a value for a property can be copied. For example, the location can be the body of the message, the local environment tree, or an exception list.

For JMS Application properties, the node provides a way to add, modify, and delete name-value pairs of application properties.

The JMSHeader node is contained in the **JMS** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Look at the following sample for more details about how to use the node:

- JMSHeader node

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

Terminals and properties

When you have put an instance of the node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. This node has no mandatory properties.

JMSHeader node terminals are described in the following table:

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the input message is routed if a failure is detected during extraction.
Out	The output terminal to which the transformed message is routed if the input message is processed successfully.

The following tables describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The JMSHeader node Description properties are described in the following table:

Property	M	C	Default	Description
Node name	No	No	JMSHeader	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The JMSHeader node JMS transport header options are described in the following table:

Property	M	C	Default	Description
JMS header options	No	Yes	Carry forward header	Options to control the JMSTransport header as a whole. Select **Carry forward header** to carry forward any values that are present in an incoming message. Select **Add header** to add a new header using the specified property values. If a header already exists, the header is modified using the specified property values. If *Inherit from header* is specified as a property value and the header does not exist, the default value for the property is used. Select **Modify header** to change an existing header using the specified property values. If a header does not exist, a new header is added first. If *Inherit from header* is specified as a property value and the header does not exist, the default value for the property is used. Select **Delete header** to delete the header, if it exists. **Note:** The **Add header** and **Modify header** options both add a header if it does not exist, and change a header if it does exist. However, the default values offered by each option differ, so use the appropriate option.

The JMSHeader node JMSHeader_Value properties are described in the following table:

Property	M	C	Default	Description	mqsiapplybaroverride command property
JMS Delivery Mode	No	Yes	Non_Persistent	Filter messages by message delivery mode: • Non_Persistent • Persistent	
JMS Message Expiration(ms)	No	Yes	0	Ask the JMS provider to keep the output JMS message for a specified time. Values are in milliseconds; the default value 0 means that the message should not expire.	
JMS Message Priority	No	Yes	4	Assign relative importance to the message. A receiving JMS client application or a JMSOutput node can use this value. JMS defines a ten-level priority value, with 0 as the lowest priority and 9 as the highest.	
JMS Correlation Identifier	No	Yes	No default value	A client can use the JMS Correlation Identifier header field to link one message with another. A typical use is to link a response message with its request message.	
JMS Reply To	No	Yes	No default value	The JMS Reply To header field contains a destination supplied by a client when a message is sent. It is the destination where a reply message should be sent to.	jmsReplyTo

The JMSHeader node Application properties are described in the following table:

Property	M	C	Default	Description
Application Properties	No	Yes		This screen is enabled only if you chose **Add header** or **Modify header** for JMS Transport header. The screen has no predefined properties; you use it to create custom properties and values. Use the property table to add new properties, or modify or delete existing properties, for the header. There is no limit to the number of properties. Each property must have a name, and a type qualifier. The type qualifier can be `Value`, `XPath`, or `Delete`. **Value** Enter a new valid value for the selected property. A null value or empty string is also considered as a valid value. **XPath** Specify a valid XPath expression. WebSphere Message Broker supports XPath definitions that start with an XPath variable such as `$Root` or `$LocalEnvironment`. Only the first occurrence is returned if there are multiple values for the XPath expression. (Examples of valid XPath expressions are: $LocalEnvironment/Host, and $Root/HTTPRequest/Content-Type). **Delete** Specify the property to be deleted from the incoming message. The value associated with the selected property is also deleted.
Clear incoming values	No	Yes	Cleared	This option, which is is enabled only if you choose **Modify header**, removes all property names and associated values from the incoming message if present.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

JMSInput node

Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider.

This topic contains the following sections:

Purpose

The JMSInput node acts as a JMS message consumer and can receive all six message types that are defined in the Java Message Service Specification, version 1.1. Messages are received by using method calls, which are described in the JMS specification.

The JMSInput node is contained in the **JMS** drawer of the palette, and is represented in the workbench by the following icon:

Using the JMSInput node in a message flow

The following sample contains a message flow in which the JMSInput node is used. This sample is an example of how to use the JMSInput node.

- JMS Nodes

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

The JMSInput node receives and propagates messages with a JMS message tree. You can set the properties of the JMSInput node to control the way in which the JMS messages are received.

The JMSInput node handles messages in the following message domains:
- BLOB
- XMLNSC
- XMLNS
- MRM

- JMSMap
- JMSStream
- MIME
- XML (this domain is deprecated; use XMLNSC)

Message flows that handle messages that are received from connections to JMS providers must always start with a JMSInput node. If you include an output node in a message flow that starts with a JMSInput node, it can be any of the supported output nodes (including user-defined output nodes); you do not need to include a JMSOutput node. However, if you do not include a JMSOutput node, you must include the JMSMQTransform node to transform the message to the format that is expected by the output node.

If you are propagating JMS messages and creating a message flow to use as a subflow, you cannot use a standard input node; you must use an instance of the JMSInput node as the first node in order to create an In terminal for the subflow.

When you use 32-bit execution groups in a default 64-bit host environment, you must set the WebSphere MQ JMS Java library paths on the MQSI_LIBPATH32. For example:

```
export MQSI_LIBPATH32=$MQSI_LIBPATH32:/usr/mqm/lib:/usr/mqm/java/lib
```

Restriction: When the JMSInput node receives publication topics, it internally restricts the message flow property Additional Instances to zero to prevent the receipt of duplicate publications.

Making the JMS provider client available to the JMS nodes

Configurable services are defined for a number of JMS providers. You can choose one of the predefined services, or you can create a new service for a new provider, or for one of the existing providers. The predefined services are listed in Configurable services properties.

- If you want to use the WebSphere MQ JMS provider, and you have installed WebSphere MQ in the default location on the broker system, the properties are already set and you do not have to make any changes.
- If you want to use the WebSphere MQ JMS provider, and you have installed WebSphere MQ in a different (non default) location, or if you want to use one of the other defined services, you must set the jarsURL property to identify the location of the service JAR files on the broker system. On Windows, the file location cannot be a mapped network drive on a remote Windows computer; the directory must be local or on a Storage Area Network (SAN) disk.

  Use the mqsireportproperties command to view the provider properties, and the mqsichangeproperties command to set or modify the properties.
- If no service is defined for your JMS provider, or if you want to create another service for an existing JMS provider, use the mqsicreateconfigurableservice command to identify the new service and to set its properties.
- When you configure the node, select the appropriate service from the list of predefined services shown for the JMS provider name property, or type in the name of your required service.
- Some JMS providers provide an alternative interface to the standard JMS specification for particular JMS API calls. In these cases, IBM supplies a Java class to interface with that proprietary API. For example, if the JMS nodes use BEA WebLogic as the JMS provider, and the nodes need to participate in a globally coordinated message flow, you must modify the configurable services

properties that are associated with that vendor. For more information, see “Configuring the broker to enable a JMS provider's proprietary API” on page 221.

- Some JMS providers, such as the BEA WebLogic provider, do not update the optional JMSXDeliveryCount field in the JMS message header; therefore, JMSInput node backout processing is not possible. To cope with any failures in the message flow, connect the Failure terminal of the JMSInput node.

Connecting the terminals

For each message that is received successfully, the JMSInput node routes the message to the Out terminal. If this action fails, the message is retried. If the retry threshold is reached, where the threshold is defined by the Backout threshold property of the node, the message is routed to the Failure terminal. You can connect nodes to the Failure terminal to handle this condition.

If an exception occurs in the failure path, the path is retried until the number of attempts is twice the Backout threshold. If that limit is exceeded, the message is put to the Backout destination.

If you have not connected nodes to the Failure terminal, the message is written to the Backout destination. If you have not specified a Backout destination, the node issues a BIP4669 error message and stops processing further input.

If processing is not resumed after you restart the broker or execution group, check the event log for a cause, such as an incorrect parser being specified in the node properties. Correct the problem and redeploy the message flow. If the message itself is not valid, remove the message from the input queue to resume processing.

If the message is caught by the JMSInput node after an exception has been generated elsewhere in the message flow, the message is routed to the Catch terminal. If you have not connected nodes to the Catch terminal, the node backs out messages for redelivery until the problem is resolved, or the Backout threshold is reached. If you do not define a Backout destination, the node issues a BIP4669 error message and stops processing further input.

Configuring for coordinated transactions

When you include a JMSInput node in a message flow, the value that you set for Transaction mode defines whether messages are received under sync point.

- If you set this property to Global, the message is received under external sync point coordination; that is, within a WebSphere MQ unit of work. Any messages that are sent subsequently by an output node in the same instance of the message flow are put under sync point, unless the output node overrides this setting explicitly.
- If you set this property to Local, the message is received under the local sync point control of the JMSInput node. Any messages that are sent subsequently by an output node in the flow are not put under local sync point, unless an individual output node specifies that the message must be put under local sync point.
- If you set this property to None, the message is not received under sync point. Any messages that are sent subsequently by an output node in the flow are not put under sync point, unless an individual output node specifies that the message must be put under sync point.

To receive messages under external sync point, you must take additional configuration steps, which need be applied only the first time that a JMSOutput or JMSInput node is deployed to the broker for a particular JMS provider.

- On distributed systems, the external sync point coordinator for the broker is WebSphere MQ. Before you deploy a message flow in which the Transaction mode property is set to Global, modify the queue manager `.ini` file to include extra definitions for each JMS provider resource manager that participates in globally-coordinated transactions.
  - Windows On Windows on x86 systems:
    1. Start WebSphere MQ Explorer.
    2. Right-click the queue manager name in the left pane and click **Properties**.
    3. Click **XA resource managers** in the left pane.
    4. Set the SwitchFile property to the following value:

       ```
       install_dir/bin/ JMSSwitch.dll
       XAOpenString=Initial Context,location JNDI,Optional_parms
       ThreadOfControl=THREAD
       ```

    For more information, see the *System Administration Guide* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.
  - Linux UNIX On Linux and UNIX systems, add a stanza to the queue manager `.ini` file for each JMS provider.

    For example:

    ```
    XAResourceManager:
    Name=Jms_Provider_Name
    SwitchFile=/install_dir/bin/ JMSSwitch.so
    XAOpenString=Initial Context,location JNDI,Optional_parms
    ThreadOfControl=THREAD
    ```

    Where:

    Name is an installation defined name that identifies a JMS provider resource manager.

    SwitchFile
    is the file system path to the JMSSwitch library that is supplied in the `bin` directory of the broker.

    XAOpenString can have the following values:

    - *Initial Context* is the value that is set in the JMSInput node property Initial context factory.
    - *location JNDI* is the value that is set in the JMSInput node property Location JNDI bindings. This value must include a supported URL prefix that has a URL handler that is available on the classpath.

    The following parameters are optional:

    - LDAP Principal matches the value that is set for the broker by using the mqsicreatebroker or mqsichangebroker commands.
    - LDAP Credentials matches the value that is set for the broker by using the mqsicreatebroker or mqsichangebroker commands.
    - Recovery Connection Factory Name is the JNDI administered connection factory that is defined in the bindings file. If a value is not specified, you must add a default value for `recoverXAQCF` to the bindings file. In either case, the Recovery Connection Factory must be defined as an XA Queue Connection Factory for the JMS provider that is associated with the Initial context factory.

The optional parameters are comma separated and are positional. Therefore, any parameters that are missing must be represented by a comma.

1. Update the Java CLASSPATH environment variable for the broker's queue manager to include a reference to `xarecovery.jar`; for example:

   install_dir`/classes/xarecovery.jar`

2. Update the Java PATH environment variable for the broker's queue manager to point to the bin directory in which the SwitchFile is located; for example:

   install_dir`/bin`

Finally, ensure that you have taken the following configuration steps:

– In the message flow, ensure that the co-ordinated property is enabled by using the Broker Archive editor.
– Ensure that each node that must be part of the XA transaction is set to the global transaction mode.
– Ensure that the service ID that is used for the broker and the queue manager is the same user ID.
– Ensure that the JNDI connection factory objects that the JMS nodes use for a global transaction are configured to be of the type MQXAConnectionFactory, MQXAQueueConnectionFactory, or MQXATopicConnectionFactory.

For more information, see the *System Administration Guide* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.

To use the same queue manager for both the broker and the JMS provider, ensure that your WebSphere MQ installation is at the minimum required level: WebSphere MQ Version 6.0 Fix Pack 1 or above is required for XA to use the same queue manager for both the broker and the provider.

- z/OS On z/OS, the external sync point manager is Resource Recovery Services (RRS). The only JMS provider that is supported on z/OS is WebSphere MQ JMS. The only transport option that is supported for WebSphere MQ JMS on z/OS is the bind option.

  sync point control for the JMS provider is managed with RRS sync point coordination of the queue manager of the broker. You do not need to modify the `.ini` file.

Terminals and properties

When you have put an instance of the JMSInput node into a message flow, you can configure it; see “Configuring a message flow node” on page 276. The properties of the node are displayed in the Properties view. All mandatory properties that do not have a default value defined are marked with an asterisk.

The terminals of the JMSInput node are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is routed if an error occurs. Even if the Validation property is set, messages that are propagated to this terminal are not validated.
Out	The output terminal to which the message is routed if it is retrieved successfully.
Catch	The output terminal to which the message is routed if an exception is generated downstream and caught by this node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the JMSInput node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, JMSInput	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the JMSInput node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Source queue	No	No	Selected	The name of the queue from which the node receives incoming messages. If the node is to read from a queue (point-to-point), select Source queue and enter the name of the source queue, which is the JMS queue that is listed in the bindings file. This property is mutually exclusive with Subscription topic.	sourceQueueName
Subscription topic	No	No	Cleared	The name of the topic to which the node is subscribed. If the node is to read from a Subscription topic (publish/subscribe), select Subscription topic and enter the name of the subscription topic. • If you select Subscription topic, the node operates in the publish/subscribe message domain only. • This property is mutually exclusive with Source queue. • The Subscription topic name must conform to the standards of the JMS provider that is being used by the node.	topic
Durable subscription ID	No	No		The identifier for a durable subscription topic. If the node is to receive publications from a durable subscription topic, enter a Durable subscription ID. • Removing a durable subscription is a separate administration task. For information about removing a durable subscription see the JMS provider documentation. • This property is valid only when a Subscription topic string has been specified.	durableSubscriptionID

The JMS Connection properties of the JMSInput node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
JMS provider name	Yes	No	WebSphere MQ	Select a JMS vendor name from the list, or enter a name of your choice. When you select a name from the list, the Initial context factory property is updated automatically with the relevant Java class. If you enter your own JMS provider name, you must also enter a value for the Initial context factory The name must match the name of a configurable service that is defined for the broker to which you deploy the message flow.	
Initial context factory	Yes	Yes	com.sun.jndi.fscontext.RefFSContextFactory	The starting point for a JNDI namespace. A JMS application uses the initial context to obtain and look up the connection factory and queue or topic objects for the JMS provider. If you select a JMS provider name from the list in JMS provider name, the Initial context factory property is updated automatically with the relevant Java class. If you enter your own JMS provider name, you must also enter a value for the Initial context factory. The default value is `com.sun.jndi.fscontext.RefFSContextFactory`, which defines the file-based Initial context factory for the WebSphere MQ JMS provider.	initialContextFactory
Location JNDI bindings	Yes	Yes		The system path or the LDAP location for the bindings file. The bindings file contains definitions for the JNDI administered objects that are used by the JMSInput node. When you enter a value for Location JNDI bindings, ensure that it complies with the following instructions: • Construct the bindings file before you deploy a message flow that contains a JMSInput node. • Do not include the file name of the bindings file in this field. • If you have specified an LDAP location that requires authentication, configure the LDAP principal (userid) and LDAP credentials (password) separately. These values are configured at broker level. For information about configuring these values, see mqsicreatebroker command and mqsichangebroker command. • The string value must include a supported URL prefix that has a URL handler that is available on the classpath. For information about constructing the JNDI administered objects bindings file, see the JMS provider documentation.	locationJndiBindings

Property	M	C	Default	Description	mqsiapplybaroverride command property
Connection factory name	Yes	Yes		The name of the connection factory that is used by the JMSInput node to create a connection to the JMS provider. This name must already exist in the bindings file. The Connection factory name can be a JMS QueueConnectionFactory or a JMS TopicConnectionFactory, but it must match the message model that is used by the node. Alternatively, you can specify the generic JMS ConnectionFactory, which can be used for both JMS queue or JMS topic destinations.	connectionFactoryName
Backout destination	No	Yes		The JMSInput node sends input messages to this destination when errors prevent the message flow from processing the message, and the message must be removed from the input destination. The backout destination name must exist in the bindings file.	backoutDestination
Backout threshold	No	Yes	0	The value that controls when a redelivered message is put to the backout destination. For example, if the value is 3, the JMS provider attempts to deliver the message to the input destination three times. After the third attempted delivery, the message is removed from the input destination and is sent to the Backout destination.	

The Input Message Parsing properties of the JMSInput node are described in the following table.

Property	M	C	Default	Description
Message domain	No	No		The domain that is used to parse the incoming message. • MRM • XMLNSC • XMLNS • JMSMap • JMSStream • MIME • BLOB • XML (this domain is deprecated; use XMLNSC) You can also specify a user-defined parser, if appropriate. The Message domain that is set on the node takes precedence except when the Message domain is set to blank on the node property. If Message domain is left blank, the JMSInput node determines the message domain in one of two ways: • By checking for the presence of data in the JMSType header value of the JMS input message • Based upon the Java Class of the JMS message For more information, see JMS message payload and appropriate parser.

Property	M	C	Default	Description
Message set	No	No		The name or identifier of the message set in which the incoming message is defined. If you are using the MRM parser or the XMLNSC parser in validating mode, select the Message set that you want to use. This list is populated with available message sets when you select MRM or XMLNSC as the Message domain. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Message type	No	No		The name of the incoming message. If you are using the MRM parser, select the message that you want from the list in Message type. This list is populated with messages that are defined in the Message set that you have selected.
Message format	No	No		The name of the physical format of the incoming message. If you are using the MRM parser, select the format of the message from the list in Message format. This list includes all of the physical formats that you have defined for this Message set.

The properties of the Parser Options for the JMSInput node are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	On Demand	This property controls when an input message is parsed. Valid values are: • On Demand • Immediate • Complete Parse timing is, by default, set to On Demand, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see "Parsing on demand" on page 1449.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the XMLNSC parser creates syntax elements in the message tree with data types taken from the XML Schema. For more information about how the XMLNSC parser operates, see "Manipulating messages in the XMLNSC domain" on page 408.
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the message data appears under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or Input Message Parsing properties Message domain is XMLNS.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in an input message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in an input message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in an input message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.

Property	M	C	Default	Description
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the input message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if Validate is None); entries that are specified in Opaque Elements are ignored if validation is enabled.

The Message Selectors properties of the JMSInput node are described in the following table. Set these properties if you need to filter messages.

Property	M	C	Default	Description
Application property	No	Yes		The message selector that filters messages according to the application property value. If the JMS provider is required to filter messages, based on message properties that are set by the originating JMS client application, enter a value for Application property, specifying both the property name and the selection conditions; for example, `OrderValue > 200`. Leave Application property blank if you do not want the input node to make a selection based on application property. For a description of how to construct the JMS message selector, see JMS message selector.
Timestamp	No	Yes		The message selector that filters messages according to the JMSTimestamp. If the JMS provider is required to filter messages that have been generated at specific times, enter a value for Timestamp, where the value is an unqualified Java millisecond time; for example, `105757642321`. Qualify the selector with operators, such as `BETWEEN` or `AND`. Leave Timestamp blank if you do not want the input node to make a selection based on the JMSTimeStamp.
Delivery mode	No	Yes	All	The message selector that filters messages according to the message delivery mode. If the JMS provider is required to filter messages based on the JMSDeliveryMode header value in the JMS messages, select an option for Delivery mode from the list: • Select Non Persistent to receive messages that are marked as non persistent by the originating JMS client application. • Select Persistent to receive messages that are marked as persistent by the originating JMS client application. • Select All to receive both persistent and non persistent messages. (This value is the default.)
Priority	No	Yes		The message selector that filters messages according to the message priority. If the JMS provider is required to filter messages based on the JMSPriority header value in the JMS message, enter a value for Priority. Valid values for Priority are from 0 (lowest) to 9 (highest); for example, enter `5` to receive messages of priority 5. You can also qualify the selector; for example, `> 4` to receive messages with a priority greater than 4, or `BETWEEN 4 AND 8` to receive messages with a priority in the range 4 to 8. Leave Priority blank if you do not want the input node to make a selection based on the JMSPriority.

Property	M	C	Default	Description
Message ID	No	Yes		The message selector that filters messages according to the message ID. If the JMS provider is required to filter messages based on the JMSMessageID header, enter a value for Message ID. Enter a specific Message ID or enter a conditional selector; for example, enter > `WMBRK123456` to return messages where the Message ID is greater than `WMBRK123456`. Leave Message ID blank if you do not want the input node to make a selection based on JMSMessageID.
Redelivered	No	Yes		If the JMS provider is required to filter messages based on the JMSRedelivered header, enter a value for Redelivered: • Enter FALSE if the input node accepts only messages that have not been redelivered by the JMS provider. • Enter TRUE if the input node accepts only messages that have been redelivered by the JMS provider. • Leave Redelivered blank if you do not want the input node to make a selection based on JMSRedelivered.
Correlation ID	No	Yes		The message selector that filters messages according to the correlation ID. If the JMS provider is required to filter messages based on the JMSCorrelationID header, enter a value for Correlation ID. Enter a specific Correlation ID or enter a conditional string; for example, `WMBRKABCDEFG` returns messages with a Correlation ID that matches this value. Leave Correlation ID blank if you do not want the input node to make a selection based on JMSCorrelationID.

The Advanced properties of the JMSInput node are described in the following table.

Property	M	C	Default	Description
Transaction mode	Yes	No	None	This property controls whether the incoming message is received under external sync point, local sync point, or out of sync point. • Select None if the incoming message is to be treated as non persistent. If you select this value, the message is received using a non-transacted JMS session that is created using the `Session.AUTO_ACKNOWLEDGE` flag. • Select Local if the JMSInput node must coordinate the commit or roll back of JMS messages that are received by the node, along with any other resources such as DB2 or WebSphere MQ that perform work within the message flow. If you select this value, the node uses a transacted JMS session. • Select Global if the JMSInput node must participate in a global message flow transaction that is managed by the broker's external sync point coordinator. The sync point coordinator is the broker's queue manager on distributed systems and RRS (Resource Recovery Services) on z/OS. If you select this value, any messages that are received by the node are globally coordinated using an XA JMS session.

The Validation properties of the JMSInput node are described in the following table. For more details, see “Validating messages” on page 204 and “Validation properties” on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are: • None • Content and Value • Content If you select Content or Content and Value, select an option from the Failure action list.	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are: • User Trace • Local Error Log • Exception (The default value) • Exception List	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

JMSMQTransform node

Use the JMSMQTransform node to transform a message with a JMS message tree into a message that has a message tree structure that is compatible with the format of messages that are produced by the WebSphere MQ JMS provider.

This topic contains the following sections:

Purpose

You can use the JMSMQTransform node to send messages to existing message flows and to work with WebSphere MQ JMS and WebSphere Message Broker publish/subscribe.

The JMSMQTransform node handles messages in all supported message domains.

The JMSMQTransform node is contained in the **JMS** drawer of the palette, and is represented in the workbench by the following icon:

Using the JMSMQTransform node in a message flow

The following sample contains a message flow in which the JMSMQTransform node is used. Look at this sample for an example of how to use the JMSMQTransform node.

- JMS Nodes

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

Terminals and properties

When you have put an instance of the JMSMQTransform node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view.

The terminals of the JMSMQTransform node are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is routed if an error occurs. Even if the Validation property is set, messages that are propagated to this terminal are not validated.
Out	The output terminal to which the message is routed if it is successfully retrieved from the JMS destination.
In	The input terminal that accepts a message for processing by the node.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The JMSMQTransform node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, JMSMQTransform	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

JMSOutput node

Use the JMSOutput node to send messages to JMS destinations.

This topic contains the following sections:

Purpose

The JMSOutput node acts as a JMS message producer, and can publish all six message types that are defined in the Java Message Service Specification, version 1.1. Messages are published by using method calls, which are described in the JMS specification.

The JMSOutput node is contained in the **JMS** drawer of the palette, and is represented in the workbench by the following icon:

Using the JMSOutput node in a message flow

The following sample contains a message flow in which the JMSOutput node is used. Look at this sample for an example of how to use the JMSOutput node.

- JMS Nodes

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

Message flows that handle messages that are received from connections to JMS providers must always start with a JMSInput node. If you include the JMSOutput node in a message flow, you do not need to include a JMSInput node; but if you do not include a JMSInput node, you must include the MQJMSTransform node to transform the message to the format that is expected by the JMSOutput node.

If you are propagating JMS messages and creating a message flow to use as a subflow, use an instance of the JMSOutput node as the last node to create an out terminal for the subflow.

Controlling the type of the JMS output message

In the JMS message tree, the JMS message type is represented by the PayloadType field of the Message_MetaData subfolder. To control the type of JMS message that is created by the JMSOutput node, use ESQL code to set the Payload value, as shown in the following example:

```
SET OutputRoot.JMSTransport.Transport_Folders.Message_MetaData.PayloadType=Payload value
```

For more information about the JMS message tree and payload values, see Representation of messages across the JMS Transport.

Sending a JMS message to a destination list

To send a JMS message to a destination list, ensure that the following conditions are met.

- Select Send to destination list in local environment on the **Basic** properties tab of the JMSOutput node.
- Set up the list in the local environment, as shown in the following example.

```
CREATE PROCEDURE CreateJMSDestinationList() BEGIN
  SET OutputLocalEnvironment.Destination.JMSDestinationList.DestinationData[1]  = 'jndi://TestDestQueue1';
  SET OutputLocalEnvironment.Destination.JMSDestinationList.DestinationData[2]  = 'jndi://TestDestQueue2';
  SET OutputLocalEnvironment.Destination.JMSDestinationList.DestinationData[3]  = 'jndi://TestDestQueue3';
END;
```

- Ensure that the message model (point-to-point or publish/subscribe) matches the model that is used by the JMSOutput node. In this case, the model is point-to-point.
- If the destination name in the list is prefixed with the string "jndi://", it indicates to the JMSOutput node that the value represents the name of a JNDI administered object, which needs to be looked up. Alternatively, if the JMS provider-specific format for a destination is known, it can be used; for example, `queue://qmgrname/queuename` for WebSphere MQ. Otherwise, the value is used to create a temporary destination.
- The items to which the JMS destination list refers represent JMS destinations that can be either JMS queues or JMS topics. These destination types must be consistent with the connection factory type that is specified in the JMSOutput node that will process the destination list. For example, a JMS queue destination can be processed by a JMS QueueConnectionFactory or a generic JMS ConnectionFactory. Similarly, a JMS topic destination can be processed by a JMS TopicConnectionFactory or a generic JMS ConnectionFactory.

For further information about using LocalEnvironment varaibles in a JMSOutput node see "Using LocalEnvironment variables with JMSOutput and JMSReply nodes" on page 1520.

Making the JMS provider client available to the JMS nodes

Configurable services are defined for a number of JMSProviders. You can choose one of the predefined services, or you can create a new service for a new provider, or for one of the existing providers. The predefined services are listed in Configurable services properties.

- If you want to use the WebSphere MQ JMS Provider, and you have installed WebSphere MQ in the default location on the broker system, the properties are already set and you do not have to make any changes.
- If you want to use the WebSphere MQ JMS Provider, and you have installed WebSphere MQ in a different (non default) location, or if you want to use one of the other defined services, you must set the jarsURL property to identify the location of the service JAR files on the broker system. On Windows, the file location cannot be a mapped network drive on a remote Windows computer; the directory must be local or on a Storage Area Network (SAN) disk.

  Use the mqsireportproperties command to view the provider properties, and the mqsichangeproperties command to set or modify the properties.
- If no service is defined for your JMS provider, or if you want to create another service for an existing JMS provider, use the mqsicreateconfigurableservice command to identify the new service and set its properties.
- When you configure the node, select the appropriate service from the list of predefined services shown for the JMS provider name property, or type in the name of your new service.
- Some JMS providers provide an alternative interface from the standard JMS specification for particular JMS API calls. In these cases, IBM supplies a Java JAR file to interface with that proprietary API. For example, if the JMS nodes use BEA WebLogic as the JMS provider, and the nodes need to participate in a globally coordinated message flow, you must modify the configurable services properties that are associated with that vendor. For more information, see "Configuring the broker to enable a JMS provider's proprietary API" on page 221.

Using the Message Destination Mode

The JMSOutput node acts as a message producer and supports the following message scenarios:

- "Sending a datagram message"
- "Sending a reply message" on page 1036
- "Sending a request message" on page 1036

For more information about how to build JMS destination lists, see "Populating Destination in the local environment tree" on page 355.

Sending a datagram message

A *datagram* is a self-contained, independent entity of data that carries sufficient information to be routed from the source to the destination computer, without reliance on earlier exchanges between the source and destination computer and the transporting network. The following instructions describe how to send a datagram message:

1. On the **Basic** tab, set the message destination depending on the message model that is being used. Set one of the following properties to a valid JNDI administered object name:
   - Publication Topic
   - Destination Queue
2. Leave the Reply To Destination field blank.

The node resolves the name of the JNDI administered object, which is supplied in either Publication Topic or Destination Queue, and sends the message to that JMS Destination.

Sending a reply message

The sender of a message might want the recipient to reply to the message. In this case, the JMSOutput message can treat the outgoing message as a reply, and route it according to the value that is obtained from the JMSReplyTo property from the request message. You can modify the value of the JMSReplyTo property in the MbMessage; for example, using a Compute node or a JavaCompute node. This action allows dynamic routing of messages from the JMSOutput node. The node sends the message to the JMS Destination name that is set in the JMSReplyTo field of the MbMessage Tree.

The JMSReplyTo value in the MbMessage Tree represents the name of the JMS Destination that is resolved from JNDI. For example:

```
queue://QM_mn2/myJMSQueue4
```

In this case, the value is the JMS-provider specific representation of a JMS Destination for the WebSphere MQ JMS provider.

If you do not want to specify a resolved JMS destination name, the JMSOutput node can also accept a JNDI administered object name in the JMSReplyTo field. However, the node must resolve an administered object name through JNDI before it can route the message to the underlying JMS Destination. In this case, the value in the JMSReplyTo field must be prefixed with the string: `jndi://`. For example:

```
jndi://jmsQ4
```

where `jmsQ4` is the name of the JNDI-administered object.

Performance might be affected when you use this method because of the need to look up the administered object in JNDI.

Sending a request message

The JMSOutput node can send a message to a JMS Destination with the expectation of a response from the message consumer that processes the request. The following instructions describe how to send a request message:

1. On the **Basic** tab, set the message destination depending on the message model that is being used. Set one of the following properties to a valid JNDI-administered object name:
   - Publication Topic
   - Destination Queue
2. The JMSReplyTo destination in the outgoing message can be derived from the JMSReplyTo field of the MbMessage Tree that is passed to the node. Alternatively, this value can be overridden by a JNDI-administered object name that is set in the Reply To Destination node property.

   To allow the JMSOutput node to set the JMSReplyTo property dynamically in the outgoing message, leave the Reply To Destination field blank on the **Basic** tab, and set the JMSReplyTo value in the MbMessage using a Compute node or a JavaCompute node.

The node looks first for a value in the JMSReplyTo field of the MbMessage. If the node finds the value, it passes this value into the JMSReplyTo field of the outgoing message. However, if the Reply To Destination field of the **Basic** tab has been specified, this value overrides anything that is set previously in the JMSReplyTo property of the outgoing message, after first resolving the name of the JNDI-administered object.

The node resolves the name of the JNDI-administered object that is supplied in either Publication Topic or Destination Queue, and sends the message to that JMS Destination.

Invoking an output message callback function

The cciOutputMessageCallback function can be registered as a callback and invoked whenever a message is sent by a JMSOutput node. See cciOutputMessageCallback.

If the user exit state is active, the cciOutputMessageCallback function is invoked for every output message that is sent successfully from a JMSOutput node where the callback is registered.

If the node provides WrittenDestination information in the LocalEnvironment tree, the callback is invoked after this information is created. See "Using LocalEnvironment variables with JMSOutput and JMSReply nodes" on page 1520.

Working with the JMS message ID

The JMS message ID is generated by the JMS provider when a message is sent by the JMSOutput node. You cannot set the message ID in the message flow, but you can use one of the following methods to obtain the generated ID after the message has been sent:

- Connect a Compute node to the Out terminal.

  Connect a Compute node to the Out terminal of a JMSOutput node and interrogate the WrittenDestination List. For more information, see "Viewing the logical message tree in trace output" on page 206.

  An entry for a JMSOutput node has the following format:

```
WrittenDestination = (
   JMS = (
      DestinationData = (
        destinationName = 'queue://jmsQueue1'
        initialContext = 'com.sun.jndi.fscontext.RefFSContextFactory'
        JMSMessageID = ID:414d512054657374514d2020202020206ab98b4520017a02'
        JMSCorrelationID = 'ABCDEFGHIJKLMNOPQRSTUVW'
  )
 )
)
```

- Configure a user exit to process an output message callback event. For more information, see "Exploiting user exits" on page 239.

Configuring for coordinated transactions

When you include a JMSOutput node in a message flow, the value that you set for Transaction Mode defines whether messages are sent under syncpoint.

- If you set the Transaction Mode to Global, the message is sent under external syncpoint coordination; that is, within a WebSphere MQ unit of work. Any messages that are sent subsequently by an output node in the same instance of the message flow are put under syncpoint, unless the output node overrides this setting explicitly.
- If you set the Transaction Mode to Local, the message is sent under the local syncpoint control of the JMSOutput node. Any messages that are sent subsequently by an output node in the flow are not put under local syncpoint, unless an individual output node specifies that the message must be put under local syncpoint.

- If you set the Transaction Mode to None, the message is not sent under syncpoint. Any messages that are sent subsequently by an output node in the flow are not put under syncpoint, unless an individual output node specifies that the message must be put under syncpoint.

When you want to send messages under external syncpoint, you must perform additional configuration steps, which need to be applied only the first time that a JMSOutput or JMSInput is deployed to the broker for a particular JMS provider:

- On distributed systems, the external syncpoint coordinator for the broker is WebSphere MQ. Before you deploy a message flow in which the Transaction Mode is set to Global, modify the queue manager `.ini` file to include extra definitions for each JMS provider resource manager that participates in globally coordinated transactions:
  - Windows On Windows on x86 systems:
    1. Start WebSphere MQ Explorer.
    2. Right-click the queue manager name in the left pane and click **Properties**.
    3. Click **XA resource managers** in the left pane.
    4. Set the SwitchFile property to the following value:

       ```
       install_dir/bin/ JMSSwitch.dll
       XAOpenString=Initial Context,location JNDI,Optional_parms
       ThreadOfControl=THREAD
       ```

       For more information, see the *System Administration Guide* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.
  - Linux UNIX On Linux and UNIX systems, add a stanza to the queue manager's `.ini` file for each JMS provider.

    For example:

    ```
    XAResourceManager:
    Name=Jms_Provider_Name
    SwitchFile=/install_dir/bin/JMSSwitch.so
    XAOpenString=Initial Context,location JNDI,Optional_parms
    ThreadOfControl=THREAD
    ```

    Where:

    Name is an installation-defined name that identifies a JMS provider resource manager.

    SwitchFile
    is the file system path to the JMSSwitch library that is supplied in the `bin` directory of the broker.

    XAOpenString can have the following values:

    - *Initial Context* is the value that is set in the JMSInput node basic property Initial context factory.
    - *location JNDI* is the value that is set in the JMSInput node basic property Location of JNDI bindings. This value must include the leading keyword, which is `file://`, `iiop://`, or `ldap://`

    The following parameters are optional:

    - LDAP Principal matches the value that is set for the broker by using the mqsicreatebroker or mqsichangebroker commands.
    - LDAP Credentials matches the value that is set for the broker by using the mqsicreatebroker or mqsichangebroker commands.

- Recovery Connection Factory Name is the JNDI administered connection factory that is defined in the bindings file. If a value is not specified, a default value for `recoverXAQCF` must be added to the bindings file. In either case, the Recovery Connection Factory must be defined as an XA Queue Connection Factory for the JMS provider that is associated with the Initial Context Factory.

The optional parameters are comma-separated and are positional. Therefore, any parameters that are missing must be represented by a comma.

1. Update the Java CLASSPATH environment variable for the broker's queue manager to include a reference to `xarecovery.jar`; for example:

   install_dir`/classes/xarecovery.jar`

2. Update the Java PATH environment variable for the broker's queue manager to point to the bin directory, which is where the switch file is located; for example:

   install_dir`/bin`

For more information, see the *System Administration Guide* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.

To use the same queue manager for both the broker and the JMS provider, ensure that your WebSphere MQ installation is at the minimum required level: Version 6.0 Fix Pack 1. WebSphere MQ Version 6.0 Fix Pack 1 or above is required for XA to use the same queue manager for both the broker and the provider.

– z/OS On z/OS, the external syncpoint manager is Resource Recovery Services (RRS). The only JMS provider that is supported on z/OS is WebSphere MQ JMS. The only transport option that is supported for WebSphere MQ JMS on z/OS is the bind option.

Syncpoint control for the JMS provider is managed with RRS syncpoint coordination of the queue manager of the broker. You do not need to modify the `.ini` file.

If the JMSOutput node uses BEA WebLogic as the JMS provider, and the nodes need to participate in a globally coordinated message flow, see "Making the JMS provider client available to the JMS nodes" on page 1034.

Connecting the terminals

Connect the In terminal of the JMSOutput node to the node from which outbound messages are routed.

Connect the Out terminal of the JMSOutput node to another node in the message flow to process the message further, to process errors, or to send the message to an additional destination.

Terminals and properties

When you have put an instance of the JMSOutput node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties that do not have a default value defined are marked with an asterisk.

The terminals of the JMSOutput node are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if an error occurs. Even if the Validation property is set, messages that are propagated to this terminal are not validated.
Out	The output terminal to which the message is routed if it is successfully retrieved from the WebSphere MQ queue.
Catch	The output terminal to which the message is routed if an exception is thrown downstream and caught by this node.

The following tables describe the node properties. The column headed M indicates whether the property is mandatory (marked with an asterisk if you must enter a value when no default is defined), the column headed C indicates whether the property is configurable (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the JMSOutput node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, JMSOutput	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the JMSOutput node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Destination Queue	No	Yes		The name of the queue to which the node publishes outgoing messages. If the JMSOutput node is to be used to send point-to-point messages, enter the Destination queue name for the JMS queue name that is listed in the bindings file.	destinationQueueName
Publication Topic	No	Yes		The name of the topic from which the node receives published messages. • If this property is configured, the node operates only in the publish/subscribe message domain. • This property is mutually exclusive with the Destination queue property. • The Publication Topic name must conform to the standards of the JMS provider that is being used by the node.	topic

Property	M	C	Default	Description	mqsiapplybaroverride command property
Reply to destination	No	Yes		The name of the JMS destination to which the receiving application must send a reply message. For a reply message to be returned to this JMS destination, the JMS destination name must be known to the domain of the JMS provider that is used by the receiving client. You can enter a JMS destination, which can be either a subscription queue or a destination topic. The default value is blank, in which case the JMS output message can be regarded as a datagram. If the field is blank, the JMSOutput node does not expect a reply from the receiving JMS client.	replyToDestination
Send to destination list in local environment	No	Yes	Cleared	When you have built a list of JMS destinations in the local environment, select this check box to use the destination list. If you do not select this check box, the node uses the configured JMS destination. If you select this check box but you have not built a list of JMS destination in the local environment, the node uses the configured JMS destination.	useDistList

The JMS Connection properties of the JMSOutput node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
JMS provider name	Yes	No	WebSphere MQ	Select a JMS vendor name from the list, or enter a name of your choice. When you select a name from the list, the Initial Context Factory property is updated automatically with the relevant Java class. If you enter your own JMS provider name, you must also enter a value for the Initial Context Factory. The name must match the name of a configurable service defined for the broker to which you deploy the message flow.	
Initial Context Factory	Yes	Yes	com.sun.jndi.fscontext.RefFSContextFactory	This property is the starting point for a JNDI namespace. A JMS application uses the initial context to obtain and look up the connection factory and queue or topic objects for the JMS provider. If you select a JMS provider name from the list in JMS provider name, the Initial Context Factory property is updated automatically with the relevant Java class. If you enter your own JMS provider name, you must also enter a value for the Initial Context Factory. The default value is `com.sun.jndi.fscontext.RefFSContextFactory`, which defines the file-based initial context factory for the WebSphere MQ JMS provider.	initialContextFactory

Property	M	C	Default	Description	mqsiapplybaroverride command property
Location JNDI Bindings	Yes	Yes		The system path or the LDAP location for the bindings file. The bindings file contains definitions for the JNDI-administered objects that are used by the JMSOutput node. When you enter a value for Location JNDI Bindings, ensure that it complies with the following instructions: • Construct the bindings file before you deploy a message flow that contains a JMSOutput node. • Do not include the file name of the bindings file in this field. • If you have specified an LDAP location that requires authentication, configure both the LDAP principal (userid) and LDAP credentials (password) separately. These values are configured at broker level. For information about configuring these values, see mqsicreatebroker command and mqsichangebroker command. • The string value must include the leading keyword, which is one of the following options: – `file://` – `iiop://` – `ldap://` For information about constructing the JNDI-administered objects bindings file, see the documentation that is supplied with the JMS provider.	locationJndiBindings
Connection Factory Name	Yes	Yes		The name of the connection factory that is used by the JMSOutput node to create a connection to the JMS provider. This name must already exist in the bindings file. The Connection factory can be a JMS QueueConnectionFactory or a JMS TopicConnectionFactory, but it must match the message model that is used by the node. Alternatively, you can specify the generic JMS ConnectionFactory, which can be used for both JMS queue or JMS topic destinations.	connectionFactoryName

The Advanced properties of the JMSOutput node are described in the following table.

Property	M	C	Default	Description
New Correlation ID	No	Yes		If the JMSOutput node is required to generate a new Correlation ID for the message, select New Correlation ID. If you leave the check box cleared, the Correlation ID of the output message is taken from the JMSCorrelationID field in the JMSTransport_Header_Values section of the message tree.

Property	M	C	Default	Description
Transaction Mode	Yes	No	None	This property controls whether the incoming message is received under syncpoint. • Select None if the outgoing message is to be treated as non persistent. If you select this value, the message is sent using a non-transacted JMS session that is created using the `Session.AUTO_ACKNOWLEDGE` flag. • Select Local if the input node that received the message must coordinate the commit or roll-back of JMS messages that have been sent by the JMSOutput node, along with any other resources, such as DB2 or WebSphere MQ, that perform work within the message flow. If you select this value, the node uses a transacted JMS session. • Select Global if the JMSOutput node must participate in a global message flow transaction that is managed by the broker's external syncpoint coordinator. The syncpoint coordinator is the broker's queue manager on distributed systems, and RRS (Resource Recovery Services) on z/OS. If you select this value, any messages that are received by the node are globally coordinated using an XA JMS session.
Delivery Mode	No	Yes	Non Persistent	This property controls the persistence mode that a JMS provider uses for a message. Valid values are: • Automatic: the mode from the input message is inherited • Persistent: the message survives if the JMS provider has a system failure • Non Persistent: the message is lost if the JMS provider has a system failure
Message Expiration (ms)	No	Yes	0	This property controls the length of time, in milliseconds, for which the JMS provider keeps the output JMS message. The default value, 0, is used to indicate that the message must not expire. Select Inherit from header or enter an integer that represents a number of milliseconds. If you select Inherit from header, the property inherits the value of the JMSExpiry field in the JMS message, which is found at the following location: `OutputRoot.JMSTransport.Transport_Folders.Header_Values.JMSExpiration`
Message Priority	No	Yes	4	This property assigns relative importance to the message and it can be used for message selection by a receiving JMS client application or a JMSOutput node. Select a value between 0 (lowest priority) and 9 (highest priority) or select Inherit from header. The default value is 4, which indicates medium priority. Priorities in the range 0 to 4 relate to typical delivery. Priorities in the range 5 to 9 relate to graduations of expedited delivery. If you select Inherit from header, the property inherits the value of the JMSPriority field in the JMS message, which is found at the following location: `OutputRoot.JMSTransport.Transport_Folders.Header_Values.JMSPriority`
Message Type	No	Yes	Determine output message type from the JMS Message Tree	Select a value from the list to configure the type of JMS message that is produced by the JMSOutput node. If you do not set a value for this property, the node assumes the output type from the metadata PayLoadType field in the JMS message tree, as indicated by the default value, Determine output message type from the JMS Message Tree. Valid values are: • Determine output message type from the JMS Message Tree • TextMessage • BytesMessage • MapMessage • StreamMessage • ObjectMessage • Base JMS message with no payload

The Validation properties of the JMSOutput node are described in the following table. For more information about Validation properties, see “Validating messages” on page 204 and “Validation properties” on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	Inherit	This property controls whether validation takes place. Valid values are None, Content, Content And Value, and Inherit.	validateMaster
Failure Action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see “Configuring monitoring event sources using monitoring properties” on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

JMSReply node

Use the JMSReply node to send messages to JMS destinations.

This topic contains the following sections:

Purpose

The JMSReply node has a similar function to the JMSOutput node, but the JMSReply node sends JMS messages only to the reply destination that is supplied in the JMSReplyTo header field of the JMS message tree. Use the JMSReply node when you want to treat a JMS message that is produced from a message flow as a reply to a JMS input message, and where you have no other routing requirements.

The JMSReply node is contained in the **JMS** drawer of the palette, and is represented in the workbench by the following icon:

Using the JMSReply node in a message flow

Consider a situation in which you create a message flow in which a JMSInput node message obtains point-to-point messages from a JMS destination called MyJMSInputQueue. The message flow updates a database using the contents of the message, then replies to a JMS destination called MyJMSReplyQueue, which is set by the generating application in the JMSReplyTo header of the input message.

In a similar scenario for the publish/subscribe message model, a JMSInput node subscribes to TopicA, and the JMSReply node publishes on the TopicB destination, which was retrieved from the JMSReplyTo header of the input message.

Calling an output message callback function

The cciOutputMessageCallback function can be registered as a callback and called whenever a message is sent by a JMSReply node. See cciOutputMessageCallback.

If the user exit state is active, the cciOutputMessageCallback function is called for every output message that is sent successfully from a JMSReply node where the callback is registered.

If the node provides WrittenDestination information in the LocalEnvironment tree, the callback is called after this information is created. See "Using LocalEnvironment variables with JMSOutput and JMSReply nodes" on page 1520.

Working with the JMS message ID

The JMS message ID is generated by the JMS provider when a message is sent by the JMSReply node. You cannot set the message ID in the message flow, but you can use one of the following methods to obtain the generated ID after the message has been sent:

- Connect a Compute node to the Out terminal.

  Connect a Compute node to the Out terminal of a JMSReply node and interrogate the WrittenDestination List. For more information, see "Viewing the logical message tree in trace output" on page 206.

  An entry for a JMSReply node has the following format:

```
WrittenDestination = (
   JMS = (
      DestinationData = (
        destinationName = 'queue://jmsQueue1'
        initialContext = 'com.sun.jndi.fscontext.RefFSContextFactory'
        JMSMessageID = ID:414d512054657374514d2020202020206ab98b4520017a02'
        JMSCorrelationID = 'ABCDEFGHIJKLMNOPQRSTUVW'
  )
 )
)
```

- Configure a user exit to process an output message callback event. For more information, see "Exploiting user exits" on page 239.

Terminals and properties

When you have put an instance of the JMSReply node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties that do not have a default value defined are marked with an asterisk.

The terminals of the JMSReply node are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is routed if an error occurs. Even if the Validation property is set, messages that are propagated to this terminal are not validated.
Out	The output terminal to which the message is routed if it is successfully retrieved from the WebSphere MQ queue.

The following tables describe the node properties. The column headed M indicates whether the property is mandatory (marked with an asterisk if you must enter a value when no default is defined), the column headed C indicates whether the property is configurable (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the JMSReply node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the JMSReply node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Send to destination list in local environment	No	Yes	Cleared	When you have built a list of JMS destinations in the local environment, select this check box to use the destination list. If you do not select this check box, the node uses the configured JMS destination. If you select this check box but you have not built a list of JMS destinations in the local environment, the node uses the configured JMS destination.	useDistList

The JMS Connection properties of the JMSReply node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
JMS provider name	Yes	No	WebSphere MQ	Select a JMS vendor name from the list, or enter a name of your choice. When you select a name from the list, the Initial Context Factory property is updated automatically with the relevant Java class. If you enter your own JMS provider name, you must also enter a value for the Initial Context Factory. The default value is WebSphere MQ.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Initial Context Factory	Yes	Yes	`com.sun.jndi.fscontext.RefFSContextFactory`	This property is the starting point for a JNDI namespace. A JMS application uses the initial context to obtain and look up the connection factory and queue or topic objects for the JMS provider. If you select a JMS provider name from the list in JMS provider name, the Initial Context Factory property is updated automatically with the relevant Java class. If you enter your own JMS provider name, you must also enter a value for the Initial Context Factory. The default value of `com.sun.jndi.fscontext.RefFSContextFactory` defines the file-based initial context factory for the WebSphere MQ JMS provider.	initialContextFactory
Location JNDI Bindings	Yes	Yes		This property specifies either the file system path or the LDAP location for the bindings file. The bindings file contains definitions for the JNDI-administered objects that are used by the JMSReply node. When you enter a value for Location JNDI Bindings, ensure that it complies with the following instructions: • Construct the bindings file before you deploy a message flow that contains a JMSReply node. • Do not include the file name of the bindings file in this field. • If you have specified an LDAP location that requires authentication, configure both the LDAP principal (userid) and LDAP credentials (password) separately. These values are configured at broker level. For information about configuring these values, refer to the mqsicreatebroker and mqsichangebroker commands. • The string value must include the leading keyword, which is one of: – `file://` – `iiop://` – `ldap://` For information about constructing the JNDI-administered objects bindings file, refer to the documentation that is supplied with the JMS provider.	locationJndiBindings

Property	M	C	Default	Description	mqsiapplybaroverride command property
Connection Factory Name	Yes	Yes		The name of the connection factory that is used by the JMSReply node to create a connection to the JMS provider. This name must already exist in the bindings file.	connectionFactoryName

The Advanced properties of the JMSReply node are described in the following table.

Property	M	C	Default	Description
New Correlation ID	No	Yes	Cleared	If the JMSReply node is required to generate a new Correlation ID for the message, select the check box. The check box is cleared by default; if you leave the check box cleared, the Correlation ID of the output message is taken from the JMSCorrelationID field in the JMSTransport_Header_Values section of the message tree.
Transaction Mode	No	No	None	This property controls whether the incoming message is received under sync point. To define the transactional characteristics of how the message is handled, select one of the following values: • Select None if the outgoing message is to be treated as non-persistent. If you select this value, the message is sent using a non-transacted JMS session that is created using the `Session.AUTO_ACKNOWLEDGE` parameter. • Select Local if the input node that receives the message should coordinate the commit or roll-back of JMS messages that have been sent by the JMSReply node, along with any other resources, such as DB2 or WebSphere MQ, that perform work within the message flow. If you select this value, the node uses a transacted JMS session. • Select Global if the JMSReply node should participate in a global message flow transaction that is managed by the broker's external sync point coordinator. The sync point coordinator is the broker's queue manager on distributed systems, and RRS (Resource Recovery Services) on z/OS. If you select this value, any messages that are received by the node are globally coordinated using an XA JMS session.
Delivery Mode	No	Yes	Automatic	This property controls the persistence mode that a JMS provider uses for a message. Valid values are: • Automatic: the mode from the input message is inherited • Persistent: the message survives if the JMS provider has a system failure • Non-persistent: the message is lost if the JMS provider has a system failure
Message Expiration (ms)	Yes	Yes	0	This property controls the length of time, in milliseconds, for which the JMS provider keeps the output JMS message. The default value, 0, is used to indicate that the message must not expire. Select Inherit from header or enter an integer that represents a number of milliseconds. If you select Inherit from header, the property inherits the value of the JMSExpiry field in the JMS message, which is found at the following location: `OutputRoot.JMSTransport.Transport_Folders.Header_Values.JMSExpiration`

Property	M	C	Default	Description
Message Priority	No	Yes	4	This property assigns relative importance to the message and it can be used for message selection by a receiving JMS client application or a JMSReply node. Select a value between 0 (lowest priority) and 9 (highest priority) or select Inherit from header. The default value is 4, which indicates medium priority. Priorities in the range 0 to 4 relate to normal delivery. Priorities in the range 5 to 9 relate to graduations of expedited delivery. If you select Inherit from header, the property inherits the value of the JMSPriority field in the JMS message, which is found at the following location: `OutputRoot.JMSTransport.Transport_Folders.Header_Values.JMSPriority`
Message type	No	Yes	TextMessage	This property controls the class of the JMS output message. The default value is TextMessage. Valid values are: • TextMessage • BytesMessage • MapMessage • StreamMessage • ObjectMessage • Base JMS message with no payload If you do not set this property, the node assumes the output type from the metadata PayLoadType field in the JMS message tree.

The Validation properties of the JMSReply node are described in the following table. Refer to "Validation properties" on page 1445 for a full description of these properties.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	Inherit	This property controls whether validation takes place. Valid values are: • None • Content and Value • Content • Inherit If a message is propagated to the Failure terminal of the node, it is not validated.	validateMaster
Failure Action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are: • User Trace • Local Error Log • Exception (default value) • Exception List	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Label node

Use the Label node to process a message that is propagated by a RouteToLabel node to dynamically determine the route that the message takes through the message flow.

This topic contains the following sections:

Purpose

Use the Label node in combination with a RouteToLabel node to route a message through the message flow based on message content. The RouteToLabel node interrogates the LocalEnvironment of the message to determine the identifier of the Label node to which the message must be routed next. You can propagate the message by coding ESQL in a Compute node, or by coding Java in a JavaCompute or user-defined node.

Precede the RouteToLabel node in the message flow with a Compute node or JavaCompute node and populate the LocalEnvironment of the message with the identifiers of one or more Label nodes that introduce the next sequence of processing for the message.

Design your message flow so that a Label node logically follows a RouteToLabel node in a message flow, but do not connect it physically to the RouteToLabel node. The connection is made by the broker, when required, according to the contents of LocalEnvironment.

The Label node provides a target for a routing decision, and does not process the message that it handles in any way. Typically, a Label node connects to a subflow that processes each message in a specific way, and either ends in an output node or in another RouteToLabel node.

The Label node can also be used in conjunction with a SOAPExtract node or as the target of a PROPAGATE statement, which is specified in a Compute or Database node.

The Label node is contained in the **Routing** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Look at the following sample to see how to use this node:

- Airline Reservations

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

Terminals and properties

When you have put an instance of the Label node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The Label node terminals are described in the following table.

Terminal	Description
Out	The output terminal to which the message is routed.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Label node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Label node Basic properties are described in the following table.

Property	M	C	Default	Description
Label Name	Yes	No		An identifier for the node. It is used as a target for a message that is routed by a RouteToLabel node. Label Name must not be the same as the name of the instance of the node itself, and it must be unique in the message flow in which it appears. The name of the instance can be modified by the workbench if the subflow, of which this Label node is a part, is embedded into another message flow.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Mapping node

Use the Mapping node to construct one or more new messages and populate them with various types of information.

This topic contains the following sections:

Purpose

You can populate the new messages with the following types of information:
- New information
- Modified information from the input message
- Information taken from a database

You can modify elements of the message body data, its associated environment, and its exception list.

When you first open or create a message map for the node, if you select This map is called from a message flow node and maps properties and message body, the headers in the input message are always copied to the output message without modification. To modify the message headers in a Mapping node, select This map is called from a message flow node and maps properties, headers, and message body. When you select this property, the map that is created allows additional elements, including WebSphere MQ, HTTP, and JMS headers, to be mapped.

These components of the output message can be defined using mappings that are based on elements of both the input message and data from an external database. You create the mappings that are associated with this node, in the mapping file that is associated with this node, by mapping inputs (message or database) to outputs. You can modify the assignments made by these mappings using supplied or user-defined functions and procedures; for example, you can convert a string value to uppercase when you assign it to the message output field.

Use the Mapping node to:
- Build a new message
- Copy messages between parsers
- Transform a message from one format to another

The Mapping node is contained in the **Transformation** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Look at the following sample to see how to use this node:

- Pager

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

Terminals and properties

When you have put an instance of the Mapping node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The Mapping node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the input message is propagated if a failure is detected during the computation. If you have selected Treat Warnings as Errors, the node propagates the message to this terminal if database warning messages are returned, even though the processing might have completed successfully.
Out	The output terminal that propagates the message following completion of the mappings.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Mapping node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Mapping node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Data Source	No	Yes		The ODBC data source name of the database that contains the tables to which you refer in the mappings that are associated with this node (identified by the Mapping Module property). This name identifies the appropriate database on the system on which this message flow is to execute. The broker connects to this database with user ID and password information that you have specified on the mqsicreatebroker, mqsichangebroker, or mqsisetdbparms command. If the name is different from the data source name used in the Mapping Editor, select the **Use data source from flow property** option on the **Override Data Source and Schema** dialog (see "Message mapping tips and restrictions" on page 600.) z/OS On z/OS systems, the broker uses the broker started task ID, or the user ID and password that are specified on the mqsisetdbparms command JCL, BIPSDBP in the customization data set <hlq>.SBIPPROC.	dataSource
Transaction	Yes	No	Automatic	The transaction mode for the node. The values are: • Automatic (the default). The message flow, of which the Mapping node is a part, is committed if it is successful; that is, the actions that you define in the mappings are performed and the message continues through the message flow. If the message flow fails, it is rolled back. If you choose Automatic, the ability to commit or rollback the action of the Mapping node on the database depends on the success or failure of the entire message flow. • Commit. To commit any uncommitted actions that are performed in this message flow on the database that is connected to this node, irrespective of the success or failure of the message flow as a whole, select Commit. The changes to the database are committed even if the message flow fails.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Mapping Routine	Yes	No	Mapping	The name of the mapping routine that contains the statements to execute against the database or the message tree. By default, the name that is assigned to the mapping routine is identical to the name of the mapping file in which the routine is defined. The default name for the file is the name of the message flow concatenated with the name of the node when you include it in the message flow (for example, `MFlow1_Mapping.msgmap` for the first Mapping node in message flow MFlow1). You cannot specify a value that includes spaces. If you click **Browse** next to this entry field, a dialog box is displayed that lists all available mapping routines that this node can access. Select the routine that you want and click **OK**; the routine name is set in Mapping Module. To work with the mapping routine that is associated with this node, double-click the node, or right-click the node and click **Open Mappings**. If the mapping routine does not exist, it is created for you with the default name in the default file. If the file exists already, you can also open file `<flow_name>_<node_name>.msgmap` in the Broker Development view. A mapping routine is specific to the type of node with which it is associated; you cannot use a mapping routine that you have developed for a Mapping node with any other node that uses mappings (for example, a DataInsert node). If you create a mapping routine, you cannot call it from any other mapping routine, although you can call it from an ESQL routine. For more information about working with mapping files, and defining their content, see “Developing message mappings” on page 546.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Mapping Mode	Yes	No	Message	The mode that is used to process information that is passed through the Mapping node. Valid values are: • Message (the default): the message is generated or passed through by the Mapping node, as modified within the node. • LocalEnvironment: the LocalEnvironment tree structure is generated or passed through by the Mapping node, as modified within the node. • LocalEnvironment And Message: the LocalEnvironment tree structure and message are generated or passed through by the Mapping node, as modified by the node. • Exception: the ExceptionList is generated or passed through by the Mapping node, as modified by the node. • Exception And Message: the ExceptionList and message are generated or passed through by the Mapping node, as modified by the node. • Exception And LocalEnvironment: the ExceptionList and LocalEnvironment tree structures are generated or passed through by the Mapping node, as modified by the node. • All: the message, ExceptionList, and LocalEnvironment are generated or passed through by the Mapping node, as modified by the node. You must set this property to reflect accurately the output message format that you need. If you select an option (or accept the default value) that does not include a particular component of the message, that component is not included in any output message that is constructed. You can choose any combination of Message, LocalEnvironment, and Exception components to be generated and modified by the Mapping node. To construct a map that propagates multiple target messages, set this property to LocalEnvironment And Message to ensure that the node executes correctly. LocalEnvironment was known as DestinationList in some previous versions; it is retained for compatibility. The Environment component of the message tree is not affected by the mode setting. Its contents, if any, are passed on from this node.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Treat Warnings as Errors	Yes	No	Cleared	For database warning messages to be treated as errors, and the node to propagate the output message to the Failure terminal, select Treat Warnings as Errors. The check box is cleared initially. When you select the check box, the node handles all positive return codes from the database as errors and generates exceptions in the same way as it does for the negative, or more serious, errors. If you do not select the check box, the node treats warnings as normal return codes, and does not raise any exceptions. The most significant warning raised is `not found`, which can be handled safely as a normal return code in most circumstances.	
Throw Exception on Database Error	Yes	No	Selected	For the broker to generate an exception when a database error is detected, select Throw Exception on Database Error. The check box is selected initially. If you clear the check box, you must handle the error in the message flow to ensure the integrity of the broker and the database. The error is ignored if you do not handle it through your own processing, because you have chosen not to invoke the default error handling by the broker. For example, you could connect the Failure terminal to an error processing subroutine.	

The parser options for the Mapping node are described in the following table.

Property	M	C	Default	Description
Use XMLNSC Compact Parser for XMLNS Domain	No	No	Cleared	If you select this check box, the outgoing MQRFH2 specifies the XMLNS instead of XMLNSC parser, allowing an external application to remain unchanged. If outgoing messages do not contain MQRFH2 headers, this property has no effect.

The Validation properties of Mapping node are described in the following table.

If a message is propagated to the Failure terminal of the node, it is not validated. These properties do not cause the input message to be validated. It is expected that, if such validation is required, the validation has already been performed by the input node or a preceding validation node. For more details about validating messages and validation properties, see “Validating messages” on page 204 and “Validation properties” on page 1445.

Property	M	C	Default	Description
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are None, Content and Value, Content, and Inherit.
Failure Action	No	No	Exception	This property controls what happens if a validation failure occurs. You can set this property only if Validate is set to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.

MQeInput node

Use the MQeInput node to receive messages from clients that connect to the broker using the WebSphere MQ Mobile Transport protocol.

Attention: Using message flows that contain MQeInput and MQeOutput nodes in Version 6.1 is deprecated. The behavior that is described here is intended only for when you are deploying from Version 6.1 to a previous version, and to provide a route for migration. Redesign your flows to remove the MQe nodes and replace them with MQ nodes that are configured to your own specifications and coordinated with your MQe gateway configuration. For more details, see Migrating a message flow that contains WebSphere MQ Everyplace nodes.

This topic contains the following sections:

Purpose

The MQeInput node receives messages that are put to a message flow from a specified bridge queue on the broker's WebSphere MQ Everyplace queue manager. The node also establishes the processing environment for the messages. You must create and configure the WebSphere MQ Everyplace queue manager before you deploy a message flow that contains this node.

Message flows that handle messages that are received across WebSphere MQ connections must always start with an MQeInput node. You can set the MQeInput node's properties to control the way in which messages are received; for example, you can indicate that a message is to be processed under transaction control.

When you deploy message flows that contain WebSphere MQ Everyplace nodes to a broker, you must deploy them to a single execution group, regardless of the number of message flows. The WebSphere MQ Everyplace nodes in the message flows must all specify the same WebSphere MQ Everyplace queue manager name. If you do not meet this restriction, an error is raised when you deploy.

The MQeInput node handles messages in the following message domains:
- MRM
- XMLNSC
- XMLNS
- JMSMap
- JMSStream
- MIME
- BLOB
- XML (this domain is deprecated; use XMLNSC)
- IDOC (this domain is deprecated; use MRM)

If you include an output node in a message flow that starts with an MQeInput node, it can be any of the supported output nodes, including user-defined output nodes; you do not need to include an MQeOutput node. You can create a message flow that receives messages from WebSphere MQ Everyplace clients and generates messages for clients that use any of the supported transports to connect to the broker, because you can configure the message flow to request the broker to provide any conversion that is necessary.

WebSphere MQ Everyplace Version 1.2.6 is used by WebSphere Message Broker. This version is compatible with later versions of WebSphere MQ Everyplace. Clients that use later versions of WebSphere MQ Everyplace (for example, Version

2.0), work correctly when connected to this node, although additional functions that are not supported in Version 1.2.6 (for example, JMS support) do not work.

Queue managers are *not* interchangeable between different versions of WebSphere MQ Everyplace. Nodes must use a queue manager that was created using Version 1.2.6. Similarly, the client must use its own level of the code when creating a queue manager.

z/OS You cannot use MQeInput nodes in message flows that you deploy to z/OS systems.

If you create a message flow to use as a subflow, you cannot use a standard input node; you must use an instance of the Input node as the first node to create an In terminal for the subflow.

If your message flow does not receive messages across WebSphere MQ connections, you can choose another supported input node.

The MQeInput node is contained in the **WebSphere MQ** drawer of the palette, and is represented in the workbench by the following icon:

Using the MQeInput node in a message flow

For an example of how this node can be used, consider a farmer who checks his fields to see how well they are irrigated. He is carrying a PDA device with WebSphere MQ Everyplace installed. He sees an area of field that requires water, and uses his PDA and a Global Satellite Navigation link to send a message to an MQeInput node. The data is manipulated using a Compute node, and a message is published by a Publication node so that a remote SCADA device can pick up the message and trigger the irrigation sprinklers. The farmer can see the water delivered to the field, minutes after sending his message.

WebSphere MQ Everyplace documentation

Find further information about WebSphere MQ Everyplace, and the properties of the node, in the WebSphere MQ Everyplace documentation on the WebSphere MQ Web page.

Configuring the MQeInput node

When you have put an instance of the MQeInput node into a message flow, you can configure it; see “Configuring a message flow node” on page 276. The properties of the node are displayed in the Properties view.

All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

Configure the MQeInput node as follows:

1. Optional: On the **Description** tab, enter a Short description, a Long description, or both. You can also rename the node on this tab.

2. On the **Default** tab, set values for the properties that describe the message domain, message set, message type, and message format that the node uses to determine how to parse the incoming message, and the default topic that is associated with the message.
   - If the incoming message has an MQRFH2 header, you do not need to set values for the Default properties because the values can be derived from the <mcd> folder in the MQRFH2 header; for example:

     ```
     <mcd><Msd>MRM</Msd><Set>DHM4UO906S001</Set><Type>receiptmsg1</Type>
     <Fmt>XML</Fmt></mcd>
     ```

     If you set values, and those values differ from those in the MQRFH2 header, the MQRFH2 header values take precedence.
   - In Message domain, select the name of the parser that you are using from the list. Choose from the following options:
     - MRM
     - XMLNSC
     - XMLNS
     - JMSMap
     - JMSStream
     - MIME
     - BLOB
     - XML (this domain is deprecated; use XMLNSC)
     - IDOC (this domain is deprecated; use MRM)

     You can also specify a user-defined parser, if appropriate.
   - If you are using the MRM or IDOC parser, or the XMLNSC parser in validating mode, select the correct message set from the list in Message set. This list is populated with available message sets when you select MRM, XMLNSC, or IDOC as the domain.
   - If you are using the MRM parser, select the correct message from the list in Message type. This list is populated with messages that are defined in the message set that you have selected.
   - If you are using the MRM or IDOC parser, select the format of the message from the list in Message format. This list includes all the physical formats that you have defined for this message set.
   - Enter the message topic in Topic. You can enter any characters as the topic name. When messages pass through the MQeInput node, they assume whatever topic name you have entered. (If you are using publish/subscribe, you can subscribe to a topic and see any messages that passed through the MQeInput node under that topic name.)
3. On the **General** tab, set the following properties:
   a. Enter the Queue name of the WebSphere MQ Everyplace bridge queue from which this input node retrieves messages. If the queue does not exist, it is created for you when the message flow is deployed to the broker.
   b. Set the level of Trace that you want for this node. If trace is available, the trace information is recorded in the file identified by Trace filename (described later in this section). Choose a level of trace:
      - None (the default). No trace output is produced, unless an unrecoverable error occurs.
      - Standard. Minimal trace output is generated to reflect the overall operations of the node.
      - Debug. Trace information is recorded at a level that helps you to debug WebSphere MQ Everyplace programs.

- Full. All available debug information is recorded to provide a full record of the node activities.

If you set the trace level to Debug or Full, you will impact the performance of WebSphere MQ Everyplace, and significant trace files can be generated. Use these options for short periods only.

c. In Trace filename, specify the name of the file to which the trace information is written. The directory structure in which the file is specified must already exist; it cannot be created during operation.

d. Select the Transaction mode to define the transactional characteristics of how this message is handled:
   - If you select Automatic, the incoming message is received under sync point if it is marked persistent; otherwise it is not. The transactionality of any derived messages that are sent subsequently by an output node is determined by the incoming persistence property, unless the output node has overridden transactionality explicitly.
   - If you select Yes, the incoming message is received under sync point. Any derived messages that are sent subsequently by an output node in the same instance of the message flow are sent transactionally, unless the output node has overridden transactionality explicitly.
   - If you select No, the incoming message is not received under sync point. Any derived messages that are sent subsequently by an output node in the message flow are sent non-transactionally, unless the output node has specified that the message must be put under sync point.

e. The Use config file check box is not selected by default; values for all properties for the MQeInput node are taken from the Properties view.

   If you select the check box, the definition of all properties is extracted from the file that is identified by Config filename (described later in this section) with the exception of the following properties:
   - The Queue name and Config filename General properties
   - All Default properties

   Use a configuration file only to specify additional properties for the node. If the properties in the Properties view are sufficient for your needs, do not select the Use config file check box.

f. If you have selected the Use config file check box, enter the full path and name of the configuration file for WebSphere MQ Everyplace in Config filename. This file must be installed on the system that supports every broker to which this message flow is deployed. If the file does not exist, an error is detected when you deploy the message flow. The default file name is `MQeConfig.ini`.

g. In Queue manager name, specify the name of the WebSphere MQ Everyplace queue manager. This queue manager is not related to the queue manager of the broker to which you deploy the message flow that contains this node.

   Only one WebSphere MQ Everyplace queue manager can be supported. Only one execution group can contain MQeInput or MQeOutput nodes. This property must therefore be set to the same value in every MQeInput node that is included in every message flow that you deploy to the same broker.

4. On the **Channel** tab, set the maximum number of channels that are supported by WebSphere MQ Everyplace in Max channels. The default value is zero, which means that there is no limit.
5. On the **Registry** tab, set the following properties:

a. Select the type of registry from the Registry type list:
   - FileRegistry. Registry and security information is provided in the Directory specified later in this section.
   - PrivateRegistry. You create the queue manager manually within WebSphere MQ Everyplace, specifying the security parameters that you need.

b. In Directory, specify the directory in which the registry file is located. This property is valid only if you have selected a Registry type of FileRegistry.

c. If you have selected a Registry type of PrivateRegistry, complete the following properties (for further details of these properties, see the WebSphere MQ Everyplace documentation):
   - Specify a PIN for the associated queue manager.
   - Specify a Certificate request PIN for authentication requests.
   - Provide a Keyring password to be used as a seed for the generation of crypto keys.
   - In Certificate host, specify the name of the certificate server that WebSphere MQ Everyplace uses for authentication.
   - In Certificate port, specify the number of the port for the certificate server that WebSphere MQ Everyplace uses for authentication.

6. On the **Listener** tab, set the following properties that define the connection type for WebSphere MQ Everyplace:

   a. In Listener type, select the adapter type to use from the list. The default value is Http; you can also select Length or History. For further details, see the WebSphere MQ Everyplace documentation.

   b. In Hostname, specify the hostname of the server. Set this property to the special value localhost or to the TCP/IP address 127.0.0.1 (the default value), both of which resolve correctly to the hostname of the server to which the message flow is deployed. You can also use any valid hostname or TCP/IP address in your network, but you must use a different message flow for each broker to which you deploy, or configure this property at deployment.

   c. In Port, specify the port number on which WebSphere MQ Everyplace is listening. If more than one MQeInput node is included in a message flow that is deployed to a single broker, each MQeInput node must specify a different number for this property. You must also ensure that the number that you specify does not conflict with other listeners on the broker system; for example, with WebSphere MQ. The default value is 8081.

   d. In Time interval (sec), specify the timeout value, in seconds, before idle channels are timed out. The default value is 300 seconds.

      Channels are persistent logical entities that last longer than a single queue manager request, and can survive network breakages, so that it might be necessary to time out channels that have been inactive for a period of time.

Connecting the terminals:

The MQeInput node routes each message that it retrieves successfully to the Out terminal; if this fails, the message is retried. If the retry timeout expires (as defined by the BackoutThreshold attribute of the input queue), the message is routed to the Failure terminal; you can connect nodes to this terminal to handle this condition. If you have not connected the Failure terminal, the message is written to the backout queue.

If the message is caught by this node after an exception has been thrown further on in the message flow, the message is routed to the Catch terminal. If you have not connected the Catch terminal, the message loops continually through the node until the problem is resolved. You must define a backout queue or a dead-letter queue (DLQ) to prevent the message looping continuously through the node.

Configuring for coordinated transactions:

When you include an MQeInput node in a message flow, the value that you set for the Transaction mode property defines whether messages are received under sync point:

- If you set the property to Yes (the default), the message is received under sync point (that is, within a WebSphere MQ unit of work). Any messages that are sent subsequently by an output node in the same instance of the message flow are put under sync point, unless the output node has overridden this explicitly.
- If you set the property to Automatic, the message is received under sync point if the incoming message is marked persistent. Otherwise, it is not. Any message that is sent subsequently by an output node is put under sync point, as determined by the incoming persistence property, unless the output node has overridden this explicitly.
- If you set the property to No, the message is not received under sync point. Any messages that are sent subsequently by an output node in the flow are not put under sync point, unless an individual output node has specified that the message must be put under sync point.

The MQeOutput node is the only output node that you can configure to override this option.

Terminals and properties

The MQeInput node terminals are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is routed if an error occurs.
Out	The output terminal to which the message is routed if it is successfully retrieved from the WebSphere MQ Everyplace queue.
Catch	The output terminal to which the message is routed if an exception is thrown downstream and caught by this node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file for deployment).

The MQeInput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	MQeInput	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The MQeInput node Default properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No		The domain that is used to parse the incoming message.
Message set	No	No		The name or identifier of the message set in which the incoming message is defined. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Message type	No	No		The name of the incoming message.
Message format	No	No		The name of the physical format of the incoming message.
Topic	No	Yes		The default topic for the input message.

The MQeInput node General properties are described in the following table.

Property	M	C	Default	Description
Queue name	Yes	Yes		The name of the WebSphere MQ Everyplace bridge queue from which this node retrieves messages for processing by this message flow.
Trace	Yes	No	None	The level of trace required for this node. Valid values are None, Standard, Debug, and Full.
Trace filename	Yes	Yes	\MQeTraceFile.trc	The name of the file to which trace records are written.
Transaction mode	Yes	No	Yes	This property controls whether the incoming message is received under sync point. Valid values are Automatic, Yes, and No.
Use config file	Yes	No	Cleared	If you select the check box, a configuration file is used for this node.
Config filename	Yes	Yes	\MQeconfig.ini	The name of the configuration file to be used if the Use config file check box is selected.
Queue manager name	Yes	Yes	ServerQM1	The name of the WebSphere MQ Everyplace queue manager.

The MQeInput node Channel properties are described in the following table.

Property	M	C	Default	Description
Max channels	Yes	No	0	The maximum number of channels that are supported by the WebSphere MQ Everyplace queue manager.

The MQeInput node Registry properties are described in the following table.

Property	M	C	Default	Description
Registry type	Yes	Yes	FileRegistry	The type of registry information to be used. Valid values are FileRegistry and PrivateRegistry.

Property	M	C	Default	Description
Directory	Yes	Yes	\ServerQM1\registry	The directory in which the registry file exists (valid only if FileRegistry is selected).
PIN	Yes	Yes		The PIN that is associated with the WebSphere MQ Everyplace queue manager (valid only if PrivateRegistry is selected).
Certificate request PIN	Yes	Yes		The PIN that is used to request authentication (valid only if PrivateRegistry is selected).
Keyring password	Yes	Yes		The password that is used to see crypto keys (valid only if PrivateRegistry is selected).
Certificate host	Yes	Yes		The name of the certificate server (valid only if PrivateRegistry is selected).
Certificate port	Yes	Yes		The port of the certificate server (valid only if PrivateRegistry is selected).

The MQeInput node Listener properties are described in the following table.

Property	M	C	Default	Description
Listener type	Yes	Yes	Http	The adapter type for the listener. Valid values are Http, Length, and History.
Hostname	Yes	Yes	127.0.0.1	The hostname of the server.
Port	Yes	Yes	8081	The port on which WebSphere MQ Everyplace listens.
Time interval (sec)	Yes	Yes	300	The WebSphere MQ Everyplace polling interval, specified in seconds.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

MQeOutput node

Use the MQeOutput node to send messages to clients that connect to the broker using the WebSphere MQ Mobile Transport protocol.

Attention: Using message flows that contain MQeInput and MQeOutput nodes in Version 6.1 is deprecated. The behavior that is described here is intended only for when you are deploying from Version 6.1 to a previous version, and to provide a route for migration. Redesign your flows to remove the MQe nodes and replace them with MQ nodes that are configured to your own specifications and coordinated with your MQe gateway configuration. For more details see Migrating a message flow that contains WebSphere MQ Everyplace nodes.

This topic contains the following sections:

Purpose

The MQeOutput node forwards messages to WebSphere MQ Everyplace queue managers. If you specify a non-local destination queue manager, ensure that there is either a route to the queue manager, or store-and-forward queue servicing for the queue manager, if it exists.

You cannot use the MQeOutput node to change the transactional characteristics of the message flow. The transactional characteristics that are set by the message flow's input node determine the transactional behavior of the flow.

z/OS You cannot use MQeOutput nodes in message flows that you deploy to z/OS systems.

If you create a message flow to use as a subflow, you cannot use a standard output node; you must use an instance of the Output node to create an out terminal for the subflow through which to propagate the message.

If you do not want your message flow to send messages to a WebSphere MQ Everyplace queue, choose another supported output node.

The MQeOutput node is contained in the **WebSphere MQ** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

For an example of how this node can be used, consider a farmer who checks his fields to see how well they are irrigated. He is carrying a PDA device with WebSphere MQ Everyplace installed. He sees that his fields are not being irrigated, and uses his PDA and a Global Satellite Navigation link to check the water flow valve, and finds that it is faulty. This information is available because the remote SCADA device that is responsible for controlling the valve publishes diagnostic messages, which are retrieved by the broker and forwarded to an MQeOutput node and on to the WebSphere MQ Everyplace client on his PDA.

WebSphere MQ Everyplace documentation

You can discover further information about WebSphere MQ Everyplace, and the properties of the node, in the WebSphere MQ Everyplace documentation on the WebSphere MQ Web page.

Connecting the terminals

Connect the In terminal to the node from which outbound messages bound are routed.

Connect the Out or Failure terminal of this node to another node in this message flow if you want to process the message further, process errors, or send the message to an additional destination. If you propagate the message, the LocalEnvironment that is associated with the message is enhanced with the following information for each destination to which the message has been put by this node:

- Queue name
- Queue manager name
- Message reply identifier (this is set to the same value as message ID)
- Message ID (from the MQMD)
- Correlation ID (from the MQMD)

These values are written in WrittenDestination within the LocalEnvironment tree structure.

If you do not connect either terminal, the LocalEnvironment tree is unchanged.

If you use aggregation in your message flows, you must use these terminals.

Terminals and properties

When you have put an instance of the MQeOutput node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The MQeOutput node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if a failure is detected when the message is put to the output queue.
Out	The output terminal to which the message is routed if it has been successfully put to the output queue, and if further processing is required within this message flow.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The MQeOutput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	MQeOutput	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The MQeOutput node Basic properties are described in the following table.

Property	M	C	Default	Description
Queue manager name	No	Yes		The name of the WebSphere MQ Everyplace queue manager to which the output queue, which is specified in Queue name, is defined. Enter a value for this property if you select Queue Name in Destination mode (on the **Advanced** tab). If you select another option for Destination mode, you do not need to set this property.
Queue name	No	Yes		The name of the WebSphere MQ Everyplace output queue to which this node puts messages. Enter a value for this property if you select Queue Name in Destination mode (on the **Advanced** tab). If you select another option for Destination mode, you do not need to set these properties.

The MQeOutput node Advanced property is described in the following table.

Property	M	C	Default	Description
Destination mode	Yes	No	Destination List	The queues to which the output message is sent: • Queue Name: the message is sent to the queue that is named in the Queue name property. The properties Queue manager name and Queue name (on the **Basic** tab) are mandatory if you select this option. • Reply To Queue: the message is sent to the queue that is named in the ReplyToQ field in the MQMD. • Destination List (the default): the message is sent to the list of queues that are named in the LocalEnvironment (also known as DestinationList) that is associated with the message.

The MQeOutput node Request properties are described in the following table.

Property	M	C	Default	Description
Request	Yes	No	Cleared	Select Request to indicate that each output message is marked in the MQMD as a request message (MQMD_REQUEST), and the message identifier field is cleared (set to MQMI_NONE) to ensure that WebSphere MQ generates a new identifier. Clear the check box to indicate that each output message is not marked as a request message. You cannot select this check box if you have selected a Destination mode of Reply To Queue.
Reply-to queue manager	No	Yes		The name of the queue manager to which the output queue, which is specified in Reply-to queue, is defined. This name is inserted into the MQMD of each output message as the reply-to queue manager. This new value overrides the current value in the MQMD.
Reply-to queue	No	Yes		The name of the reply-to queue to which to put a reply to this request. This name is inserted into the MQMD of each output message as the reply-to queue. This new value overrides the current value in the MQMD.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

MQGet node

Use the MQGet node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and the MQI and AMI application programming interfaces.

You can also use the MQGet node to retrieve messages that were previously placed in a WebSphere MQ message queue that is defined to the broker queue manager.

This topic contains the following sections:

The topic uses the following terms:

input message
A message that enters the In terminal of the MQGet node.

queue message
A message that the MQGet node reads from the queue.

Purpose

The MQGet node reads a message from a specified queue, and establishes the processing environment for the message. If appropriate, you can define the input queue as a WebSphere MQ clustered queue or shared queue.

You can use an MQGet node anywhere in a message flow, unlike an MQInput node, which you can use only as the first node in a message flow. The output message tree from an MQGet node is constructed by combining the input tree with the result tree from the MQGET call. You can set the properties of the MQGet node to control the way in which messages are received; for example, you can indicate that a message is to be processed under transaction control, or you can request that, when the result tree is being created, data conversion is performed on receipt of every input message.

The MQGet node handles messages in the following message domains:
- MRM
- XMLNSC
- DataObject
- XMLNS
- JMSMap
- JMSStream
- MIME
- BLOB
- XML (this domain is deprecated; use XMLNSC)
- IDOC (this domain is deprecated; use MRM)

The MQGet node is contained in the **WebSphere MQ** drawer of the palette, and is represented in the workbench by the following icon:

Using the MQGet node in a message flow

For information about how to use the MQGet node in a message flow, see “A request-response scenario that uses an MQGet node” on page 230.

Look at the following sample to see how to browse messages with the MQGet node:

- Browsing WebSphere MQ Queues

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

Configuring the MQGet node

When you have put an instance of the MQGet node into a message flow, you can configure it; for more information, see “Configuring a message flow node” on page 276. The properties of the node are displayed in the Properties view.

All mandatory properties for which you must enter a value (those properties that do not have a default value defined) are marked with an asterisk.

Configuring for coordinated transactions

When you include an MQGet node in a message flow, the value that you set for Transaction mode defines whether messages are received under sync point.

- If you set the property to Yes (the default), the queue message is received under sync point (that is, in a WebSphere MQ unit of work). Any messages that an output node in the same instance of the message flow sends later are put under sync point, unless the output node, or any other subsequent node, overrides this setting explicitly.
- If you set the property to Automatic, the queue message is received under sync point if the incoming message is marked as persistent. Otherwise, it is not received under sync point. Any message that is sent later by an output node is put under sync point, as determined by the incoming persistence property, unless the output node, or any other subsequent node, overrides this setting explicitly.
- If you set the property to No, the queue message is not received under sync point. Any messages that are sent later by an output node in the message flow are not put under sync point, unless an individual output node, or any other subsequent node, specifies that the message must be put under sync point.

If you set the Browse only property, the value that you set for the Transaction mode property is ignored because a message cannot be browsed under sync point. However, any derived messages that are propagated later by an output node in the same instance of the message flow follow the behavior that is described previously in accordance with the specified Transaction mode value.

Overriding node properties during message processing

When you include and configure an MQGet node in a message flow, you might want to override its properties under some conditions. For example, you might want to read from a queue that is identified in another part of the message, or that is retrieved from a database record.

To override the values that you set for the MQGet node properties to achieve a more dynamic way to process messages, include a Compute or JavaCompute node in your message flow before the MQGet node. Configure this node to create an output message, and add fields to the local environment tree to define new values for the properties that you want to change.

For example, add a Compute node into the flow and define a new queue name for the MQGet node to read for messages, by including the following ESQL statement:

```
SET LocalEnvironment.MQ.GET.QueueName = 'new_queue';
```

Use `LocalEnvironment.MQ.GET.` as the correlation name for all fields that relate to the MQGet node.

Connecting the terminals

Connect the Out, Warning, Failure, and No Message output terminals of this node to another node in the message flow to process the message further, process errors, or send the message to an additional destination.

The completion code (CC) that is generated by the MQGET call controls what is propagated to each of the output terminals.

- If the MQGET call is successful, the MQGet node routes each parsed output message to the Out terminal.
- If the MQGET call fails, but with a CC that indicates a warning, an unparsed output message is propagated to the Warning terminal.
- If the MQGET call fails with a CC more severe than a warning, the input message is propagated to the Failure terminal.
- If the MQGET call fails with a reason code of MQRC_NO_MSG_AVAILABLE, the output message is propagated (without a result body) to the No Message terminal. The output message that is propagated to the No Message terminal is constructed from the input message only, according to the values of the Generate mode, Copy message, and Copy local environment properties.
- If you do not connect the Out, Warning, or No Message terminals to another node in the message flow, any message that is propagated to those terminals is discarded.
- If you do not connect the Failure terminal to another node in the message flow, the broker generates an exception when a message is propagated to that terminal.

For more information, see “Connecting failure terminals” on page 247.

Terminals and properties

The terminals of the MQGet node are described in the following table.

Terminal	Description
In	The input terminal that accepts the message that is being processed by the message flow.
Warning	The output terminal to which the output tree is propagated if an error (with a CC that indicates a warning) occurs in the node while trying to get a message from the queue. The MQMD part of the message is parsed, but the rest of the message is an unparsed BLOB element. The warning is discarded if the terminal is not connected, and there is no output propagation from the node at all.
Failure	The output terminal to which the input message is routed if an error (with a CC that indicates an error that is more severe than a warning) occurs in the node while trying to get a message from the queue.
Out	The output terminal to which the message is routed if it is retrieved successfully from the WebSphere MQ queue.
No Message	The output terminal to which the input message is routed if no message is available on the queue. The output message that is propagated to the No Message terminal is constructed from the input message only, according to the values of the Generate mode, Copy message, and Copy local environment properties.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the MQGet node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, MQGet	The name of the node.
Short description	No	No	Blank	A brief description of the node.
Long description	No	No	Blank	Text that describes the purpose of the node in the message flow.

The Basic properties of the MQGet node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Queue name	Yes	Yes	None	The name of the WebSphere MQ message queue from which this node retrieves messages. You must predefine this queue to the queue manager that hosts the broker on which the message flow is deployed. If this queue is not a valid queue, the node generates an exception, and the input message is propagated to the Failure terminal.	queueName

The Input Message Parsing properties of the MQGet node are described in the following table.

If the queue message has an MQRFH2 header, you do not have to set values for the Input Message Parsing properties, because the values can be derived from the <mcd> folder in the MQRFH2 header; for example:

```
<mcd><Msd>MRM</Msd><Set>DHM4UO906S001</Set><Type>receiptmsg1</Type>
<Fmt>XML</Fmt></mcd>
```

If you set values, and the values differ from those values in the MQRFH2 header, the values in the MQRFH2 header take precedence.

Property	M	C	Default	Description
Message domain	No	No	BLOB	The domain that is used to parse the queue message. If the MQRFH2 header does not supply the Message domain value, then you can select a value from the list. If you do not select a value, the default value is BLOB. You can also specify a user-defined parser, if appropriate.
Message set	No	No	None	The name or identifier of the message set in which the queue message is defined. If you are using the MRM or IDOC parser, or the XMLNSC parser in validating mode, select the Message set that you want to use. If you set this property, then later update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Message type	No	No	None	The name of the queue message. If you are using the MRM parser, select the correct message from the list in Message type. This list is populated with messages that are defined in the Message set that you have selected.
Message format	No	No	None	The name of the physical format of the queue message. If you are using the MRM or IDOC parser, select the format of the message from the list in Message format. This list includes all the physical formats that you have defined for this Message set. If you set the Message domain property to DataObject, you can set this property to XML or SAP ALE IDoc. Set this property to SAP ALE IDoc when you need to parse a bit stream from an external source and generate a message tree.

The Parser Options properties of the MQGet node are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	On Demand	This property controls when the queue message is parsed. Valid values are On Demand, Immediate, and Complete. By default, this property is set to On Demand, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see “Parsing on demand” on page 1449.
Use MQRFH2C compact parser for MQRFH2 header	No	No	Cleared	This property controls whether the MQRFH2C compact parser, instead of the MQRFH2 parser, is used for MQRFH2 headers. Select Use MQRFH2C compact parser for MQRFH2 header if you want the MQRFH2C parser to be used. By default, this check box is cleared, which means that the compact parser is not used.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the XMLNSC parser creates syntax elements in the message tree with data types taken from the XML schema. You can select this property only if you set the Validate property on the **Validation** tab to Content or Content and Value. For more information about XMLNSC, see “Manipulating messages in the XMLNSC domain” on page 408.

Property	M	C	Default	Description
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the message data appears under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or Input Message Parsing properties Message domain is XMLNS.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in the queue message message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in the queue message message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in the queue message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the queue message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if Validate is None); entries that are specified in Opaque Elements are ignored if validation is enabled.

The Advanced properties of the MQGet node are described in the following table.

Property	M	C	Default	Description
Transaction mode	No	No	Yes	This property controls whether the incoming message is received under sync point. Select a value for Transaction mode from the list to define the transactional characteristics of how this message is handled: • If you select Automatic, the queue message is received under sync point if it is marked as persistent. If the message is not marked as persistent, it is not received under sync point. The persistence or non-persistence of the input message determines the transactionality of any derived messages that are later propagated by an output node, unless the output node, or any other subsequent node in the message flow, overrides the transactionality explicitly. • If you select Yes, the queue message is received under sync point. Any derived messages that are later propagated by an output node in the same instance of the message flow are sent transactionally, unless the output node, or any other subsequent node in the message flow, overrides the transactionality explicitly. • If you select No, the queue message is not received under sync point. Any derived messages that are later propagated by an output node in the same instance of the message flow are sent non-transactionally, unless the output node, or any other subsequent node in the message flow, has specified that the messages must be put under sync point.

Property	M	C	Default	Description
Generate mode	No	No	Message	This property controls which parts of the message from the input tree are copied. Select a value for Generate mode from the list to define which components of the output message are generated in the MQGet node, and which components are taken from the input message. • If you select None, all the components of the message from the input tree are propagated unchanged. • If you select Message (the default), a new Message component is created by the node, but the local environment, environment, and exception list components from the input tree are propagated unchanged. • If you select LocalEnvironment, a new local environment component is created by the node, but the message, environment, and exception list components from the input tree are propagated unchanged. • If you select Message and LocalEnvironment, new message and local environment components are created by the node, but the environment and exception list components from the input tree are propagated unchanged.
Copy message	No	No	None	This property controls which parts of the message from the input tree are copied. If you have set Generate mode to either Message or Message And LocalEnvironment, select a value for Copy message from the list to define which parts of the message are generated in the MQGet node, and which parts are taken from the input message. • If you select None (the default), no part of the input message from the input tree is propagated. • If you select Copy Headers, the headers from the input message in the input tree are copied to the output message. • If you select Copy Entire Message, the entire input message from the input tree is copied to the output message.
Copy local environment	No	No	Copy Entire LocalEnvironment	This property controls how the local environment is copied to the output message. If you have set Generate mode to either LocalEnvironment or Message And LocalEnvironment, select a value for Copy Local Environment from the list to define which parts of the local environment are generated in the MQGet node, and which parts are taken from the input message. • If you select Copy Entire LocalEnvironment (the default), at each node in the message flow, a new copy of the local environment is created in the tree, and it is populated with the contents of the local environment from the preceding node. Therefore, if a node changes the local environment, the upstream nodes do not see those changes because they have their own copies. This behavior might be an issue if you are using a FlowOrder node, or if you use the propagate command on a Compute node. The entire local environment that is defined in the input message is copied to the output message. • If you select None, each node does not generate its own copy of the local environment, but it uses the local environment that is passed to it by the previous node. Therefore, if a node changes the local environment, those changes are seen by the upstream nodes.

Property	M	C	Default	Description
Wait interval (ms)	Yes	No	1000	The maximum time, in milliseconds, to wait for the queue message to be obtained from the message queue. Provide a value for the Wait interval (ms) property to specify how many milliseconds to wait for a message to be received from the MQGET call. If you do not provide a value, the default value of 1000 milliseconds is used.
Minimum message buffer size (KB)	Yes	No	4	The minimum size, in kilobytes, of the get buffer. The minimum value of this property is 1. Provide a value for this property to specify the size of the initial buffer for the MQGET call. The buffer expands automatically to accept a message of any size, but if messages are likely to be large, specify a suitable value to reduce the frequency of the buffer being resized. If you do not provide a value, the size of the buffer is 4 KB.

The Request properties of the MQGet node are described in the following table.

Property	M	C	Default	Description
Input MQMD location	No	No	InputRoot.MQMD	The location in the input message assembly where the MQMD that is to be used for the MQGET can be found. The default location is InputRoot.MQMD.
Input MQ parameters location	No	No	InputLocalEnvironment.MQ.GET	The location in the input message assembly where the WebSphere MQ parameters (for example, the initial buffer size and the MQGMO overrides) can be found. The default location is InputLocalEnvironment.MQ.GET.
Get by correlation ID	No	No	Cleared	If you select this check box, only messages that have the specified correlation ID are retrieved. If you select Get by correlation ID, the `CorrelId` field of the message to be retrieved must match the `CorrelId` field in the Input MQMD location. By default, this check box is cleared. Setting the `CorrelId` field to MQCI_NONE has the same effect as not selecting Get by correlation ID.
Get by message ID	No	No	Cleared	If you select this check box, only messages that have the specified message ID are retrieved. If you select Get by message ID, the `MsgId` field of the message to be retrieved must match the `MsgId` field in the Input MQMD location. By default, this check box is cleared.
Use all input MQMD fields	No	No	Cleared	If you select Use all input MQMD fields, all MQMD fields at the Input MQMD location are used to retrieve the message. If an MQMD bit stream is present at the Input MQMD location, all fields in the bit stream are used. Make sure that the MQMD of the message to be retrieved matches these fields. By default, this check box is cleared.

Property	M	C	Default	Description
Browse only	No	No	Cleared	This property controls whether a message is removed from the queue when it is read. If this check box is selected, the message is not removed from the queue when it is read. Select Browse only to specify that the message must be retained on the queue when it is read.

The Result properties of the MQGet node are described in the following table. Set these properties to determine how the results of the MQGET call are handled.

Property	M	C	Default	Description
Output data location	No	No	OutputRoot	This property specifies where the output data is placed. If you leave the field blank, OutputRoot is used as a default. Enter the start location in the output message tree at which the parsed elements from the bit string of the queue message are stored. All elements at this location are deleted, and the default behavior is to replace the input tree message with the queue message. You can enter any valid ESQL field reference (this reference can include expressions), including new field references to create a node in the message tree for inserting the response into the message that is propagated from the input tree. For example, `OutputRoot.XMLNS.ABC.DEF` and `Environment.GotReply` are valid field references. For more detailed information, see "A request-response scenario that uses an MQGet node" on page 230. When the queue message bit string is parsed to create the contents of the message tree, the message properties that you have specified as the Input Message Parsing properties of the node are used.
Result data location	No	No	ResultRoot	This property specifies which subtree (of the queue message) to use. If you leave this field blank, ResultRoot is used as a default, and the whole queue message is used. If, for example, ResultRoot.MQMD.ReplyToQ is specified, only that subtree is used. Set this property to control which subtree of the queue message is placed in the output message. If, for example, you want only the MQMD from the queue message, use `ResultRoot.MQMD`; this subtree is then placed at the location specified by Output data location.

Property	M	C	Default	Description
Output MQ parameters location	No	No	OutputLocalEnvironment.MQ.GET	This property specifies where the output WebSphere MQ parameters are located. If you leave this field blank, OutputLocalEnvironment.MQ.GET is used as a default. Set Generate mode to include LocalEnvironment to ensure that the updated values are visible in downstream nodes. The default location is OutputLocalEnvironment.MQ.GET. Set this property to control where the CC (completion code), the RC (reason code), the Browsed indicator, and any other WebSphere MQ parameters (for example, the MQMD that is used by the MQGET call) are placed in the output tree.
Warning data location	No	No	OutputRoot	This property specifies where the output data is placed if MQGET returns a warning code. If you leave this field blank, OutputRoot is used as a default. Set this property to control where the queue message is placed when the MQGET call returns a warning code. You can enter any valid ESQL field reference (see the description of the Output data location property). The data that is placed at this location is always the complete result tree, with the body as a BLOB element. Result data location is not used for warning data.
Include message contents in output message assembly	No	No	Selected	This property specifies that no result or warning data is required for the output message assembly. If you select this check box, the node gets or browses the message on the queue without completely reading or parsing its contents. If you select Include message contents in output message assembly, the message contents are not guaranteed to be included in the output tree because this inclusion depends on other node properties, such as the Generate mode property. Clear Include message contents in output message assembly to specify that no result or warning data is required for the output message assembly. This action gets or browses the message on the queue without reading or parsing its contents.

The Validation properties of the MQGet node are described in the following table. For a full description of these properties, see "Validation properties" on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are None, Content, Content and Value, and Inherit.	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

MQHeader node

Use the MQHeader node to add, modify, or delete MQ Message Descriptor (MQMD) and MQ Dead Letter Header (MQDLH) headers.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow"
- "Terminals and properties"

Purpose

You can add or remove a whole header, or you can change only certain fields in a header. You can set these fields to a fixed value, or to a value specified by an XPath expression to access a value in one of the message trees. XPath is used to provide a valid location from which a value for a property can be copied. For example, the location can be the body of the message, the local environment tree, or an exception list.

The MQHeader node is contained in the **WebSphere MQ** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Look at the following sample for more details about how to use the node:
- MQHeader node

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

Terminals and properties

When you have put an instance of the node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. This node has no mandatory properties.

MQHeader node terminals are described in the following table:

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the input message is routed if a failure is detected.
Out	The output terminal to which the transformed message is routed if the input message is processed successfully.

The following tables describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The MQHeader node Description properties are described in the following table:

Property	M	C	Default	Description
Node name	No	No	MQHeader	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The MQ Message Descriptor properties are described in the following table: Refer to *WebSphere MQ Application Programming Reference* and *WebSphere MQ Application Programming Guide* for full details of each of the MQ property and its supported values.

Property	M	C	Default	Description	mqsiapplybaroverride command property
MQMD header options	No	No	Carry forward header	Options to control the MQMD as a whole. Select Carry forward header to carry forward any values that are present in an incoming message. Select Add header to add a new header using the specified property values. If a header already exists, the header is modified using the specified property values. If Inherit from header is specified as a property value and the header does not exist, the default value for the property is used. Select Modify header to change an existing header using the specified property values. If a header does not exist, a new header is added first. If Inherit from header is specified as a property value and the header does not exist, the default value for the property is used. Select Delete header to delete the header, if it exists. **Note:** The Add header and Modify header options both add a header if it does not exist, and change a header if it does exist. However, the default values offered by each option differ, so use the appropriate option.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Coded Character Set Identifer	No	No	MQCCSI_Q_MGR	The character set identifier of character data in the message. A sample set of custom literals for EBCIDIC and other common Unicode values is given here: MQCCSI_INTL_EBCIDIC : 500 MQCCSI_US_EBCIDIC : 037 MQCCSI_UNICODE_1200 : 1200 MQCCSI_UNICODE_1208 : 1208 MQCCSI_UNICODE_13488 : 13488 MQCCSI_UNICODE_17584 : 17584 Refer to the *WebSphere MQ Application Programming Reference* and *WebSphere MQ Application Programming Guide* for full details.	
Format	No	No	MQFMT_NONE	A name that the sender of the message can use to indicate to the receiver the nature of the data in the message.	
Version Number	No	No	MQMD_VERSION_1	The version ID of the MQMD message.	
Message Type	No	No	MQMT_DATAGRAM	The message type.	
Message Expiry	No	No	MQEI_UNLIMITED	A period of time expressed in tenths of a second, set by the application that puts the message. The message becomes eligible to be discarded if it has not been removed from the destination queue before this period of time elapses.	
Feedback or Reason Code	No	No	MQFB_NONE	Used with a message of type MQMT_REPORT to indicate the nature of the report, and meaningful only with that type of message.	
Message Priority	No	No	MQPRI_PRIORITY_AS_Q_DEF	Message priority. 0 is the lowest value, and 9 is the highest. Custom display literals are as follows: MQPRI_PRIORITY_HIGH : 9 MQPRI_PRIORITY_8 : 8 MQPRI_PRIORITY_7 : 7 MQPRI_PRIORITY_6 : 6 MQPRI_PRIORITY_5 : 5 MQPRI_PRIORITY_MEDIUM : 4 MQPRI_PRIORITY_3 : 3 MQPRI_PRIORITY_2 : 2 MQPRI_PRIORITY_1 : 1 MQPRI_PRIORITY_LOW : 0	
Message Persistence	No	No	MQPER_PERSISTENCE_AS_Q_DEF	Indicates whether the message survives system failures and restarts of the queue manager.	
Message Identifier	No	No	MQMI_NONE	A string that is used to distinguish one message from another.	
Correlation Identifier	No	No	MQCI_NONE	A string that the application can use to relate one message to another, or to relate the message to other work that the application is performing.	
Reply To Queue	No	Yes	<No default value>	The message queue to which the application that issued the get request for the message should send Reply and Report messages.	mqmdReplyToQ
Reply To Queue Manager	No	Yes	<No default value>	The queue manager to which the reply message or report message should be sent.	mqmdReplyToQMgr

The Report properties are described in the following table:

Property	M	C	Default	Description
Inherit from header	No	No	Selected	This property is enabled only when the Modify header option is selected. Select this field to inherit any MQMD report property value that is present in an incoming message.
Exception	No	No	No default value	A type of MQ report message. Exception report message is generated.
Expiration	No	No	No default value	A type of MQ report message. Expiration report message is generated.
Confirm on arrival	No	No	No default value	A type of MQ report message. Confirm on arrival report message is generated.
Confirm on delivery	No	No	No default value	A type of MQ report message. Confirm on delivery report message is generated.
Notification	No	No	No default value	A type of MQ report Message. Action notification report message is generated.

The MQDLH header properties are described in the following table:

Property	M	C	Default	Description	mqsiapplybaroverride command property
MQDLH header options	No	No	Carry forward the header	Options to control the MQMD as a whole. Select Carry forward header to carry forward any values that are present in an incoming message. Select Add header to add a new header using the specified property values. If a header already exists, the header is modified using the specified property values. If Inherit from header is specified as a property value and the header does not exist, the default value for the property is used. Select Modify header to change an existing header using the specified property values. If a header does not exist, a new header is added first. If *Inherit from header* is specified as a property value and the header does not exist, the default value for the property is used. Select Delete header to delete the header, if it exists. **Note:** The Add header and Modify header options both add a header if it does not exist, and change a header if it does exist. However, the default values offered by each option differ, so use the appropriate option.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Coded Character Set Identifer	No	No	MQCCSI_UNDEFINED	The character set identifier of character data in the message.	
Format	No	No	MQFMT_NONE	A name that the sender of the message can use to indicate to the receiver the nature of the data in the message.	
Reason	No	No	MQRC_NONE	A code that indicates why the message is sent to the dead letter queue (DLQ).	
Destination Queue Name	No	Yes	No default value	The name of the destination queue.	mqdlhDestQName
Destination Queue Manager Name	No	Yes	No default value	The name of the destination queue manager.	mqdlhDestQMgrName
Save dead letter queue	No	No	Selected	If selected, the dead letter queue name is stored in the local environment.	
Save source queue	No	No	Selected	If selected, the original source queue name is stored in the local environment.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

MQInput node

Use the MQInput node to receive messages from clients that connect to the broker by using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

This topic contains the following sections:

Purpose

The MQInput node receives a message from a WebSphere MQ message queue that is defined on the broker's queue manager. The node uses MQGET to read a

message from a specified queue, and establishes the processing environment for the message. If appropriate, you can define the input queue as a WebSphere MQ clustered queue or shared queue.

Message flows that handle messages that are received across WebSphere MQ connections must always start with an MQInput node. You can set the properties of the MQInput node to control the way in which messages are received, by causing appropriate MQGET options to be set. For example, you can indicate that a message is to be processed under transaction control. You can also request that data conversion is performed on receipt of every input message.

The MQInput node handles messages in the following message domains:
- MRM
- XMLNSC
- DataObject
- XMLNS
- JMSMap
- JMSStream
- MIME
- BLOB
- XML (this domain is deprecated; use XMLNSC)
- IDOC (this domain is deprecated; use MRM)

If you include an output node in a message flow that starts with an MQInput node, the output node can be any one of the supported output nodes, including user-defined output nodes; you do not have to include an MQOutput node. You can create a message flow that receives messages from WebSphere MQ clients and generates messages for clients that use any of the supported transports to connect to the broker, because you can configure the message flow to request that the broker provides any conversion that is necessary.

If you create a message flow to use as a subflow, you cannot use a standard input node; you must use an Input node as the first node to create an In terminal for the subflow.

If your message flow does not receive messages across WebSphere MQ connections, you can choose one of the supported input nodes.

The MQInput node is contained in the **WebSphere MQ** drawer of the palette, and is represented in the workbench by the following icon:

Using the MQInput node in a message flow

Look at the following samples to see how to use the MQInput node:
- Pager
- Airline Reservations
- Error Handler
- Aggregation
- JMS Nodes
- Large Messaging

- Message Routing
- Scribble
- Soccer Results
- Timeout Processing
- Video Rental
- XSL Transform
- Browsing WebSphere MQ Queues

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

Connecting the terminals

The MQInput node routes each message that it retrieves successfully to the Out terminal. If this action fails, the message is tried again. If the backout count is exceeded (as defined by the BackoutThreshold attribute of the input queue), the message is routed to the Failure terminal; you can connect nodes to this terminal to handle this condition. If you have not connected the Failure terminal, the message is written to the backout queue.

If the message is caught by this node after an exception has been thrown further on in the message flow, the message is routed to the Catch terminal. If you have not connected the Catch terminal, the message loops continually through the node until the problem is resolved.

You must define a backout queue or a dead-letter queue (DLQ) to prevent the message from looping continually through the node.

Configuring for coordinated transactions

When you include an MQInput node in a message flow, the value that you set for Transaction mode defines whether messages are received under sync point:

- If you set the property to Automatic, the message is received under sync point if the incoming message is marked as persistent; otherwise, it is not received under sync point. Any message that is sent subsequently by an output node is put under sync point, as determined by the incoming persistence property, unless the output node has overridden this explicitly.
- If you set the property to Yes (the default), the message is received under sync point; that is, within a WebSphere MQ unit of work. Any messages that are sent subsequently by an output node in the same instance of the message flow are put under sync point, unless the output node has overridden this explicitly.
- If you set the property to No, the message is not received under sync point. Any messages that are sent subsequently by an output node in the message flow are not put under sync point, unless an individual output node has specified that the message must be put under sync point.

The MQOutput node is the only output node that you can configure to override this option.

If you have set the Browse Only property, the value that is set for the Transaction mode property is ignored because a message cannot be browsed under sync point. However, any derived messages that are propagated subsequently by an output node in the same instance of the message flow follow the behavior that is described previously in accordance with the specified Transaction mode value.

MQGET buffer size

The MQGET buffer size is implemented internally by the broker and you cannot change it. The following description is provided for information only. You must not rely on it when you develop your message flows, because the internal implementation might change.

When the MQInput node initializes, it sets the size of the default buffer for the first MQGET to 4 KB. The MQInput node monitors the size of messages and increases or reduces the size of the buffer:

1. If an MQGET fails because the message is larger than the size of the buffer, the node immediately increases the size of the buffer to accommodate the message, issues the MQGET again, and zeros a message count.
2. When 10 messages have been counted since the increase in the size of the buffer, the node compares the size of the largest of the 10 messages with the size of the buffer. If the size of the largest message is less than 75% of the size of the buffer, the buffer is reduced to the size of the largest of the 10 messages. If an MQGET fails during the 10 messages because the message is larger than the size of the buffer, the node takes action 1.

For example, if the first message that the node receives is 20 MB, and the next 10 messages are each 14 MB, the size of the buffer is increased from 4 KB to 20 MB and remains at 20 MB for 10 messages. However, after the 10th message the size of the buffer is reduced to 14 MB.

Terminals and properties

When you have put an MQInput node into a message flow, you can configure the node; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties that do not have a default value defined are marked with an asterisk.

The terminals of the MQInput node are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is routed if an error occurs. Even if the Validation property is set, messages propagated to this terminal are not validated.
Out	The output terminal to which the message is routed if it is successfully retrieved from the WebSphere MQ queue.
Catch	The output terminal to which the message is routed if an exception is thrown downstream and caught by this node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the MQInput node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, MQInput	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the MQInput node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Queue name	Yes	Yes		The name of the WebSphere MQ input queue from which this node retrieves messages (using MQGET) for processing by this message flow. You must predefine this WebSphere MQ queue to the queue manager that hosts the broker to which the message flow is deployed.	queueName

The Input Message Parsing properties of the MQInput node are described in the following table. Set values for these properties to describe the message domain, message set, message type, and message format that the node uses to determine how to parse the incoming message.

If the incoming message has an MQRFH2 header, you do not have to set values for the Input Message Parsing properties because the values are derived from the <mcd> folder in the MQRFH2 header; for example:

```
<mcd><Msd>MRM</Msd><Set>DHM4UO906S001</Set><Type>receiptmsg1</Type>
<Fmt>XML</Fmt></mcd>
```

If you set values, and those values differ from those in the MQRFH2 header, and the <Msd> element is a valid domain, the values in the MQRFH2 header take precedence.

Property	M	C	Default	Description
Message domain	No	No	BLOB	The domain that is used to parse the incoming message. If no MQRFH2 header exists to supply the value for the Message domain, you can select the property value from the list. You can either select an option or leave the property blank, in which case the default that is used is BLOB. The following options are available: • MRM • XMLNSC • DataObject • XMLNS • JMSMap • JMSStream • MIME • BLOB • XML (this domain is deprecated; use XMLNSC) • IDOC (this domain is deprecated; use MRM) You can also specify a user-defined parser, if appropriate.

Property	M	C	Default	Description
Message set	No	No		If you use the MRM or IDOC parser, or the XMLNSC parser in validating mode, select the name or identifier of the message set in which the incoming message is defined. The list of message sets consists of those that are available when you select MRM, XMLNSC, or IDOC as the domain. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Message type	No	No		If you use the MRM parser, select the type of message from the list. This list is populated with messages that are defined in the message set that you have selected.
Message format	No	No		If you are using the MRM or IDOC parser, select the physical format of the incoming message from the list. This list includes all the physical formats that you have defined for this message set. If you set the Message domain property to DataObject, you can set this property to XML or SAP ALE IDoc. Set this property to SAP ALE IDoc when you need to parse a bit stream from an external source and generate a message tree.

The properties of the Parser Options for the MQInput node are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	On Demand	This property controls when an input message is parsed. Valid values are On Demand, Immediate, and Complete. Parse timing is, by default, set to On Demand, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see “Parsing on demand” on page 1449.
Use MQRFH2C compact parser for MQRFH2 header	No	No	Cleared	This property controls whether the MQRFH2C compact parser, instead of the MQRFH2 parser, is used for MQRFH2 headers.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the XMLNSC parser creates syntax elements in the message tree with data types taken from the XML Schema. You can select this property only if you set the Validate property on the **Validation** tab to Content or Content and Value.
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	This property controls whether the XMLNSC compact parser is used for messages in the XMLNS domain. If you set this property, the message data appears under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or the Input Message Parsing property Message domain is XMLNS. For more information, see “Manipulating messages in the XMLNSC domain” on page 408.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in an input message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in an input message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.

Property	M	C	Default	Description
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in an input message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the input message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if Validate is None); entries that are specified in Opaque Elements are ignored if validation is enabled.

The Advanced properties of the MQInput node are described in the following table. Set these properties to determine how the message is processed, such as its transactional characteristics. Many of these properties map to options on the MQGET call.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Transaction mode	Yes	No	Yes	This property controls whether the incoming message is received under sync point. Valid values are Automatic, Yes, and No. • If you select Automatic, the incoming message is received under sync point if it is marked persistent, otherwise it is not received under sync point. The transactionality of any derived messages that are sent subsequently by an output node is determined by the incoming persistence property, unless the output node has overridden transactionality explicitly. • If you select Yes, the incoming message is received under sync point. Any derived messages that are sent subsequently by an output node in the same instance of the message flow are sent transactionally, unless the output node has overridden transactionality explicitly. • If you select No, the incoming message is not received under sync point. Any derived messages that are sent subsequently by an output node in the message flow are sent non-transactionally, unless the output node has specified that the messages must be put under sync point.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Order mode	Yes	No	Default	The order in which messages are retrieved from the input queue and processed. Valid values are Default, By User ID, and By Queue Order. This property has an effect only if the message flow property Additional instances on the **Instances** tab, is set to greater than zero; that is, if multiple threads read the input queue. Valid values are: • Default. Messages are retrieved in the order that is defined by the queue attributes, but this order is not guaranteed because the messages are processed by the message flow. • By User ID. Messages that have the same UserIdentifier in the MQMD are retrieved and processed in the order that is defined by the queue attributes; this order is guaranteed to be preserved when the messages are processed. A message that is associated with a particular UserIdentifier that is being processed by one thread, is completely processed before the same thread, or another thread, can start to process another message with the same UserIdentifier. No other ordering is guaranteed to be preserved. • By Queue Order. Messages are retrieved and processed by this node in the order that is defined by the queue attributes; this order is guaranteed to be preserved when the messages are processed. This behavior is identical to the behavior that is exhibited if the message flow property Additional instances is set to zero. However, if you set Order mode to By Queue Order then redeploy the message flow, additional instances that are already running are not released. Therefore, when you set Order mode to By Queue Order, either stop and restart the message flow, or run the mqsireload command for the execution group after you redeploy the flow. For more details about using this option, see “Receiving messages in a WebSphere MQ message group” on page 706.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Logical order	Yes	No	Selected	If you select this check box, messages are received in logical order, as defined by WebSphere MQ. This option maps to the MQGMO_LOGICAL_ORDER option of the MQGMO of the MQI. If you clear the check box, messages that are sent as part of a group are not received in a predetermined order. If a broker expects to receive messages in groups, and you have not selected this check box, either the order of the input messages is not significant, or you must design the message flow to process them appropriately. You must also select Commit by message group if you want message processing to be committed only after the final message of a group has been received and processed. More information about the options to which this property maps is available in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online. For more details about using this option, see "Receiving messages in a WebSphere MQ message group" on page 706.	
All messages available	Yes	No	Cleared	Select All messages available if you want message retrieval and processing to be done only when all messages in a single group are available. This property maps to the MQGMO_ALL_MSGS_AVAILABLE option of the MQGMO of the MQI. Clear this check box if message retrieval does not depend on all of the messages in a group being available before processing starts. More information about the options to which this property maps is available in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Match message ID	No	No		A message ID that must match the message ID in the MQMD of the incoming message. Enter a message identifier if you want the input node to receive only messages that contain a matching message identifier value in the MsgId field of the MQMD. Enter an even number of hexadecimal digits (characters `0 to 9`, `A to F`, and `a to f` are valid) up to a maximum of 48 digits. If the matching message identifier that you enter is shorter than the size of the MsgId field, Match message ID is padded on the right with `X'00'` characters. This property maps to the MQMO_MATCH_MSG_ID option of the MQGMO of the MQI. Leave this property blank if you do not want the input node to check that the message ID matches. More information about the options to which this property maps is available in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.	
Match correlation ID	No	No		A correlation ID that must match the correlation ID in the MQMD of the incoming message. Enter a message identifier if you want the input node to receive only messages that contain a matching correlation identifier value in the CorrelId field of the MQMD. Enter an even number of hexadecimal digits (characters `0 to 9`, `A to F`, and `a to f` are valid) up to a maximum of 48 digits. If the ID that you enter is shorter than the size of the CorrelId field, it is padded on the right with `X'00'` characters. This property maps to the MQMO_MATCH_CORREL_ID option of the MQGMO of the MQI. Leave this property blank if you do not want the input node to check that the CorrelID matches. More information about the options to which this property maps is available in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Convert	Yes	No	Cleared	If you select this check box, WebSphere MQ converts data in the message to be received, in conformance with the CodedCharSetId and Encoding values set in the MQMD. Select Convert if you want WebSphere MQ to perform data conversion on the message when it is retrieved from the queue. This option is useful if you are handling messages in the BLOB domain, or are using a user-defined parser. Do not select this option if you are parsing messages with the XML or MRM domains, because the parser does the conversion. WebSphere MQ converts the incoming message to the encoding and coded character set that is specified in the MQMD that the input node supplies on the MQGET call to retrieve the message from the input queue. The broker uses the MQGMO_CONVERT option on the MQGET call; typical rules for WebSphere MQ conversion apply. The values that you specify in the Convert encoding and Convert coded character set ID options are used in the **MsgDesc Encoding** and **CCSID** fields in the MQGET call. WebSphere MQ can convert the incoming message only if the MQMD **Format** field is a built-in WebSphere MQ value that identifies character data, or if a data conversion exit exists in WebSphere MQ. This property maps to the MQGMO_CONVERT option of the MQGMO of the MQI. Clear the check box if you do not want WebSphere MQ to convert the message. If you select this check box, you can also specify values for the Convert encoding and Convert coded character set ID properties. For more information about WebSphere MQ data conversion, and why you might choose to use this option, see the *Application Programming Guide* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Convert encoding	No	No		The representation used for numeric values in the message data, expressed as an integer value. This property is valid only if you have selected Convert. Enter the number that represents the encoding to which you want to convert numeric data in the message body. Valid values include: • Linux Windows 546 for DOS, all Windows systems, and Linux on x86 • Linux UNIX 273 for Linux on Linux on POWER, Linux on System z, and all UNIX systems • z/OS 785 for z/OS messages that use Binary Packed Decimals, and 273 for messages that use IEEE floating point numbers The encoding is used only in the following circumstances: • If a user-defined WebSphere MQ data conversion exit uses the encoding. • If the built-in WebSphere MQ conversion exit uses the encoding to convert messages in any of the predefined WebSphere MQ formats. If you specify an incorrect value, no conversion is done. For further information about the values that you can specify for Convert encoding, see "Converting data with message flows" on page 165 and the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.	
Convert coded character set ID	No	No		The coded character set identifier of character data in the message data, expressed as an integer value. This property is valid only if you have selected Convert. Enter the value that represents the character set identifier to which you want to convert character data in the message body. If you specify an incorrect value, no conversion is done. For further information about the values that you can specify for Convert coded character set ID, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Commit by message group	Yes	No	Cleared	This property controls when a transaction is committed when processing messages that are part of a message group. If you select the check box, the transaction is committed when the message group has been processed. If you leave this check box cleared, a commit is performed after each message has been propagated completely through the message flow. This property is relevant only if you have selected Logical order. Set the Order mode property to By Queue Order if the messages in a group must be retrieved and processed in the order in which they are displayed on the queue.	
z/OS serialization token	No	No		On z/OS only: A user-defined token for serialized application support. The value that is specified must conform to the rules for a valid ConnTag in the WebSphere MQ MQCNO structure. Enter a serialization token if you want to use the serialized access to shared resources that is provided by WebSphere MQ. The value that you provide for the serialization token must conform to the rules described in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online. For more information about serialization and queue sharing on z/OS, see the *Concepts and Planning Guide* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.	serializationToken
Topic	No	Yes		The default topic for the input message. You can associate a message with a publish/subscribe topic by using this property. You can enter any characters as the topic name. When messages pass through the MQInput node, they take on whatever topic name you have entered. (If you are using publish/subscribe, you can subscribe to a topic and see any messages that passed through the MQInput node and were published under that topic name.) If the incoming message has an MQRFH2 header, you do not have to set a value for the Topic property because the value can be obtained from the `<psc>` folder in the MQRFH2 header; for example: `<psc><Topic>stockquote</Topic></psc>` If you set a Topic property value, and that value differs from the <Topic> value in the MQRFH2 header, the value in the MQRFH2 header takes precedence.	topicProperty
Browse Only	No	No	Cleared	This property controls whether a message is removed from the queue when it is read. If you select this check box, the message is not removed from the queue when it is read. If you select this option, `OutputLocalEnvironment.MQ.GET.Browsed` is set to true when a message is propagated to the output terminal of the MQInput node.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Reset browse timeout (ms)	Yes	Yes	-1	The time, in milliseconds, between the last eligible message being browsed on the input queue and the browse being reset to the beginning of the queue. The default value of -1 indicates that the browse is not reset.	

The Validation properties of the MQInput node are described in the following table. Set these properties if you want the parser to validate the body of messages against the Message set. (If a message is propagated to the Failure terminal of the node, it is not validated.)

For more details, see "Validating messages" on page 204 and "Validation properties" on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are None, Content, and Content and Value.	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Security properties of the MQInput node are described in the following table. Set values for these properties to control the extraction of an identity from a message (when a security profile is associated with the node). For more information about these properties, see Identity, Configuring the extraction of an identity, Message flow security , and Setting up message flow security.

Property	M	C	Default	Description
Identity token type	No	No	None	This property specifies the type of identity token present in the incoming message. Valid values are: • Transport Default • Username • Username + Password • X.509 Certificate If this property is not specified, the identity is retrieved from the transport headers and the type is set to Username.
Identity token location	No	No	None	This property specifies where, in the message, the identity can be found. The location is specified as an ESQL field reference or XPath expression. If this property is not specified, the identity is retrieved from the MQMD.UserIdentifier transport header.
Identity password location	No	No	None	This property specifies where, in the message, the password can be found. The location is specified as an ESQL field reference or XPath expression. If it is not specified, the password is not set. This property can be set only if Identity token type is set to Username + Password.
Identity IssuedBy location	No	No	None	This property specifies a string or path expression that describes the issuer of the identity. The location is specified as an ESQL field reference or XPath expression. If this property is not specified, the MQMD.PutApplName value is used. If you leave the Identity issuedBy location field blank and the MQMD.PutApplName is also blank, the string `MQ` is used.

Property	M	C	Default	Description
Treat security exceptions as normal exceptions	No	No	False	This property specifies whether to treat security exceptions (such as "Access Denied") as normal exceptions, and propagate them to the Failure terminal (if wired). This property is turned off by default, which ensures that security exceptions cause the message to be backed out even if the Failure terminal is wired.

The Instances properties of the MQInput node are described in the following table. Set values for these properties to control the additional instances that are available for a node. For a full description of these properties, see "Configurable message flow properties" on page 1324.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Additional instances pool	No	Yes	Use Pool Associated with Message Flow	The pool from which additional instances are obtained. • If you select Use Pool Associated with Message Flow, additional instances are obtained from the message flow value. • If you select Use Pool Associated with Node, additional instances are allocated from the node's additional instances based on the number specified in the Additional instances property.	componentLevel
Additional instances	No	Yes	0	The number of additional instances that the node can start if the Additional instances pool property is set to Use Pool Associated with Node. By default, no additional instances are given to the node.	additionalInstances

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

MQJMSTransform node

Use the MQJMSTransform node to receive messages that have a WebSphere MQ JMS provider message tree format, and transform them into a format that is compatible with messages that are to be sent to JMS destinations.

This topic contains the following sections:

Purpose

Use the MQJMSTransform node to send messages to existing message flows and to interoperate with WebSphere MQ JMS and WebSphere Message Broker publish/subscribe.

The JMSMQTransform node handles messages in all supported message domains.

The MQJMSTransform node is contained in the **JMS** drawer of the palette, and is represented in the workbench by the following icon:

Using the MQJMSTransform node in a message flow

The following sample contains a message flow in which the MQJMSTransform node is used. Look at this sample for an example of how to use the MQJMSTransform node.

- JMS Nodes

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

Terminals and properties

When you have put an instance of the MQJMSTransform node into a message flow, you can configure it; see “Configuring a message flow node” on page 276. The properties of the node are displayed in the Properties view.

The terminals of the MQJMSTransform node are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is routed if an error occurs. Even if the Validation property is set, messages that are propagated to this terminal are not validated.
Out	The output terminal to which the message is routed if it is successfully retrieved from the WebSphere MQ queue.
In	The input terminal that accepts a message for processing by the node.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The MQJMSTransform node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, MQJMSTransform	The name of the node.

Property	M	C	Default	Description
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

MQOptimizedFlow node

Use the MQOptimizedFlow node to provide a high-performance publish/subscribe message flow. The node supports publishers and subscribers that use Java Message Service (JMS) application programming interfaces and the WebSphere MQ Enterprise Transport.

Restriction: z/OS You cannot use an MQOptimizedFlow node in message flows that you deploy to z/OS systems.

This topic contains the following sections:

Purpose

Use the MQOptimizedFlow node to replace a publish/subscribe message flow that consists of an MQInput node that is connected to a Publication node and that uses the Java Message Service (JMS) over WebSphere MQ transport.

Use the MQOptimizedFlow node to improve performance, particularly where a single publisher produces a persistent publication for a single subscriber.

The MQOptimizedFlow node is contained in the **WebSphere MQ** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Use an MQOptimizedFlow node in a message flow to publish a persistent JMS message to a single subscriber.

The MQOptimizedFlow node has no terminals, so you cannot connect it to any other message flow node.

Terminals and properties

When you have put an instance of the MQOptimizedFlow node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The MQOptimizedFlow node has no terminals. It is a complete message flow and you cannot connect it to other message flow nodes to extend the message processing.

The following tables describe the node properties. The column headed M indicates whether the property is mandatory; that is, whether you must enter a value if no default value is defined; an asterisk next to the name of the property denotes this. The column headed C indicates whether the property is configurable; that is, whether you can change the value in the BAR file.

The MQOptimizedFlow node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	MQOptimizedFlow	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The MQOptimizedFlow node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Queue name	Yes	Yes	None	The name of the WebSphere MQ input queue from which this node retrieves messages for processing by this message flow.	queueName

The MQOptimizedFlow node Advanced properties are described in the following table.

Property	M	C	Default	Description
Transaction mode	Yes	No	Yes	This property controls whether the incoming message is received under syncpoint. Valid values are Automatic, Yes, and No.

MQOutput node

Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

This topic contains the following sections:

Purpose

The MQOutput node delivers an output message from a message flow to a WebSphere MQ queue. The node uses MQPUT to put the message to the destination queue or queues that you specify.

If appropriate, define the queue as a WebSphere MQ clustered queue or shared queue. When you use a WebSphere MQ clustered queue, leave the Queue Manager Name empty.

You can configure the MQOutput node to put a message to a specific WebSphere MQ queue that is defined on any queue manager that is accessible by the queue manager for the broker, or to the destinations identified in the local environment that is associated with the message.

Set other properties to control the way in which messages are sent, by causing appropriate MQPUT options to be set; for example, you can request that a message is processed under transaction control. You can also specify that WebSphere MQ can, if appropriate, break the message into segments in the queue manager.

If you create a message flow to use as a subflow, you cannot use a standard output node; use an instance of the Output node to create an Out terminal for the subflow through which to propagate the message.

If you do not want your message flow to send messages to a WebSphere MQ queue, choose another supported output node.

The MQOutput node checks for the presence of an MQMD (WebSphere MQ message descriptor) header in the message tree. If no MQMD header is present, the MQOutput node creates one in the message tree, and populates it with MQMD default properties. If an MQMD header is found, the MQOutput node checks that it is an WebSphere MQ type header; if it is not, the Message Context property is set to Default. The MQOutput node treats any other transport headers in the message tree as data. If such headers are not required as part of the message body, use a transformation node to remove them from the message tree before the MQOutput node. If the message tree is sourced from JMS, you can use a JMSMQTransform node to construct a message tree compatible with MQ. For further details, see JMS message transformation.

The MQOutput node is contained in the **WebSphere MQ** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Look at the following samples to see how to use this node:
- Pager
- Airline Reservations
- Error Handler

- Aggregation
- Large Messaging
- Message Routing
- Timeout Processing
- Video Rental
- XSL Transform

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

For an example of how to use this node, assume that you have written a publishing application that publishes stock updates on a regular basis. The application sends the messages to the broker on an MQInput node, and the message flow makes the publications available to multiple subscribers through a Publication node. You configure a Compute node to create a new output message whenever one particular stock is changed, and connect this node to an MQOutput node to record each price change for this stock.

Working with WrittenDestination data

After the message has been put, the WrittenDestination folder in the local environment is updated with the destination information. WrittenDestination data for an MQOutput node has the following format:

```
WrittenDestination = (
   MQ  = (
      z`DestinationData = (
         queueName        = 'OUT'
         queueManagerName = 'MYQUEUEMANAGER'
         replyIdentifier = X'4d...2e'
         msgId           = X'3c...2c'
         correlId        = X'2a...00'
         GroupId         = X'3a...00'
      )
   )
)
```

Connecting the terminals

Connect the In terminal to the node from which outbound messages bound are routed.

Connect the Out or Failure terminal of this node to another node in this message flow to process the message further, process errors, or send the message to an additional destination.

If you use aggregation in your message flows, you must use the output terminals.

Contents of the WebSphere MQ reply message

The:

- Values of the following fields in MQMD are set, irrespective of the settings you make:

  ```
  MQMD.Report = 0;
  MQMD.PutApplType = MQAT_BROKER;
  MQMD.PutDate = Taken from current Timestamp
  MQMD.PutTime = Taken from current Timestamp
  MQMD.PutApplName = MsgTree.MQMD.ReplyToQMgr (first 28 chars)
  ```

- Values of the following fields are set from the values in the `Root.MQMD` folder:

```
MQMD.Version
MQMD.Format
MQMD.Priority
MQMD.Persistence
MQMD.Expiry
MQMD.Encoding
MQMD.CodedCharSetId
MQMD.GroupId
MQMD.MsgSeqNumber
MQMD.Offset
MQMD.MsgFlags
MQMD.OriginalLength
```

- Following values in MQMD are set conditionally, based on values in the MQOutput node and the `Root.MQMD` folder:

```
IF MsgTree.MQMD.MsgType = MQMT_REQUEST THEN
  MQMD.MsgType = MQMT_REPLY;
IF Nodes Message Context is Default, PassAll or PassIdentity THEN
  MQMD.UserIdentifer = MsgTree.MQMD.UserIdentifier;
IF MsgTree.MQMD.Report contains MQRO_PASS_CORREL_ID THEN
  MQMD.CorrelId = MsgTree.MQMD.CorrelId;
ELSE
  MQMD.CorrelId = MsgTree.MQMD.MsgId;
IF MsgTree.MQMD.Report contains MQRO_PASS_MSG_ID THEN
  MQMD.MsgId = MsgTree.MQMD.MsgId;
ELSE
  MQMD.MsgId = MQMI_NONE;
```

- Value of the `MQMD.Persistence` field is set based on the `Persistence` mode on the MQOutput node.

When the output MQMD structure has been constructed, the `Message Context` on the MQOutput node is ignored, and the behavior is as `set All`.

The values that are overridden, are done only in the output MQMD structure; no updates are made to the MQMD folder in the message tree.

Configuring for coordinated transactions

When you define an MQOutput node, the option that you select for the Transaction Mode property defines whether the message is written under sync point:

- If you select Yes, the message is written under sync point (that is, within a WebSphere MQ unit of work).
- If you select Automatic (the default), the message is written under sync point if the incoming input message is marked as persistent.
- If you select No, the message is not written under sync point.

Another property of the MQOutput node, Persistence Mode, defines whether the output message is marked as persistent when it is put to the output queue:

- If you select Yes, the message is marked as persistent.
- If you select Automatic (the default), the message persistence is determined from the properties of the incoming message, as set in the MQMD.
- If you select No, the message is not marked as persistent.
- If you select As Defined for Queue, the message persistence is set as defined in the WebSphere MQ queue by the MQOutput node specifying the MQPER_PERSISTENCE_AS_Q_DEF option in the MQMD.

Terminals and properties

When you have put an instance of the MQOutput node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view.

The MQOutput node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if a failure is detected when the message is put to the output queue.
Out	The output terminal to which the message is routed if it has been successfully put to the output queue, and if further processing is required within this message flow.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file for deployment).

The MQOutput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, MQOutput	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The MQOutput node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Queue Manager Name	No	Yes		Enter the name of the WebSphere MQ queue manager to which this output queue (which is specified by the Queue Name property) is defined. If you set the Destination Mode property to Queue Name, you must set this property. If you set Destination Mode to another value, this property is ignored.	queueManagerName
Queue Name	No	Yes		To send the output message to a single destination queue that is defined by this node, enter the name of the WebSphere MQ output queue to which the message flow sends messages. If you set the Destination Mode property to Queue Name, you must specify a value for the Queue Name property. If you set Destination Mode to another value, this property is ignored.	queueName

The MQOutput node Advanced properties are described in the following table. These properties define the transactional control for the message and the way in which the message is put to the queue. Many of these properties map to options on the MQPUT call.

Property	M	C	Default	Description
Destination Mode	Yes	No	Queue Name	The queues to which the output message is sent. • If you select Queue Name (the default), the message is sent to the queue that is named in the Queue Name property. If you select this option, you must set the Queue Manager Name and Queue Name properties. • If you select Reply To Queue, the message is sent to the queue that is named in the ReplyToQ field in the MQMD. When you select this value, the MQOutput node constructs a WebSphere MQ reply message. For more information about the settings that are used by the MQOutput node and the `Root.MQMD` folder in this situation, see "Contents of the WebSphere MQ reply message" on page 1102. • If you select Destination List, the message is sent to the list of queues that are named in the local environment that is associated with the message. The data that you have provided is used in the DestinationData subtree of the local environment. For more information about the DestinationData subtree, see "Data types for elements in the MQ DestinationData subtree" on page 1519. For more information about destination lists, see "Creating destination lists" on page 201.
Transaction Mode	Yes	No	Automatic	This property controls whether the message is put transactionally. • If you select Automatic (the default), the message transactionality is derived from the way that it was specified at the input node. • If you select Yes, the message is put transactionally. • If you select No, the message is put non-transactionally. For more information, see "Configuring for coordinated transactions" on page 1103.
Persistence Mode	Yes	No	Automatic	This property controls whether the message is put persistently. • If you select Automatic (the default), the persistence is as specified in the incoming message. • If you select Yes, the message is put persistently. • If you select No, the message is put non-persistently. • If you select As Defined for Queue, the message persistence is set as defined for the WebSphere MQ queue.
New Message ID	Yes	No	Cleared	If you select this check box, WebSphere MQ generates a new message identifier to replace the contents of the **MsgId** field in the MQMD. This property maps to the MQPMO_NEW_MSG_ID option of the MQPMO of the MQI. Clear the check box if you do not want to generate a new ID. A new message ID is still generated if you select the Request property on the **Request** tab. For more information about the options to which this property maps, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.

Property	M	C	Default	Description
New Correlation ID	Yes	No	Cleared	If you select this check box, WebSphere MQ generates a new correlation identifier to replace the contents of the CorrelId field in the MQMD. This property maps to the MQPMO_NEW_CORREL_ID option of the MQPMO of the MQI. Clear the check box if you do not want to generate a new ID. For more information about the options to which this property maps, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.
Segmentation Allowed	Yes	No	Cleared	If you select this check box, WebSphere MQ breaks the message into segments in the queue manager. Clear the check box if you do not want segmentation to occur. For more information about message segmentation, see "Sending message segments in a WebSphere MQ message" on page 709. For more information about the options to which this property maps, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.
Message Context	Yes	No	Pass All	This property controls how origin context is handled. • Pass All maps to the MQPMO_PASS_ALL_CONTEXT option of the MQPMO of the MQI. • Pass Identity maps to the MQPMO_PASS_IDENTITY_CONTEXT option of the MQPMO of the MQI. • Set All maps to the MQPMO_SET_ALL_CONTEXT option of the MQPMO of the MQI. • Set Identity maps to the MQPMO_SET_IDENTITY_CONTEXT option of the MQPMO of the MQI. • Default maps to the MQPMO_DEFAULT_CONTEXT option of the MQPMO of the MQI. • None maps to the MQPMO_NO_CONTEXT option of the MQPMO of the MQI. When a security profile is associated with the node and is configured to perform identity propagation, the chosen context can be overridden to ensure that the outgoing identity is set. For more information about the options to which these properties map, see the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.
Alternate User Authority	Yes	No	Cleared	If you select this check box, alternate authority is used when the output message is put and the MQOO_ALTERNATE_USER_AUTHORITY option is set in the open options (MQOO) of the MQI. If you select this check box, this option is specified when the queue is opened for output. The alternative user information is retrieved from the context information in the message. Clear the check box if you do not want to specify alternative user authority. If you clear the check box, the broker service user ID is used when the message is put.

The MQOutput node Request properties are described in the following table. These properties define the characteristics of each output message that is generated.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Request	Yes	No	Cleared	If you select the check box, each output message in the MQMD is generated as a request message (MQMT_REQUEST), and the message identifier field is cleared (and set to MQMI_NONE) so that WebSphere MQ generates a new identifier. Clear the check box to indicate that each output message is not marked as a request message. If you have set Destination Mode to Reply To Queue, you cannot select this check box. A new message identifier is generated even if the New Message ID check box is not selected on the **Advanced** tab.	
Reply-to Queue Manager	No	Yes		The name of the WebSphere MQ queue manager to which the output queue, which is specified in Reply-to Queue, is defined. This name is inserted into the MQMD of each output message as the reply-to queue manager.	replyToQMgr
Reply-to Queue	No	Yes		The name of the WebSphere MQ queue to which to put a reply to this request. This name is inserted into the MQMD of each output message as the reply-to queue.	replyToQ

The Validation properties of the MQOutput node are described in the following table.

For a full description of these properties, see "Validation properties" on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	Inherit	This property controls whether validation takes place. Valid values are None, Content and Value, Content, and Inherit.	validateMaster
Failure Action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

MQReply node

Use the MQReply node to send a response to the originator of the input message.

This topic contains the following sections:

Purpose

The MQReply node is a specialized form of the MQOutput node that puts the output message to the WebSphere MQ queue that is identified by the ReplyToQ field of the input message header. If appropriate, you can define the queue as a WebSphere MQ clustered queue or shared queue.

The MQReply node uses the options that are set in the Report field in the MQMD. By default (if no options are set), the MQReply node generates a new MsgId field in the reply message, and copies the message ID from the input message to the CorrelId field in the reply message. If the receiving application expects other values in these fields, ensure that the application that puts the message to the message flow input queue sets the required report options, or that you set the appropriate options in the MQMD during message processing in the message flow; for example, use a Compute node to set the Report options in the message.

More information about the Report field is available in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.

The MQReply node is contained in the **WebSphere MQ** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Look at the following sample to see how to use this node:
- Airline Reservations

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

You can use this node when you receive an order from a customer. When the order message is processed, a response is sent to the customer acknowledging receipt of the order and providing a possible date for delivery.

Working with data in the WrittenDestination folder

After the message has been put to the reply queue, the WrittenDestination folder in the local environment tree is updated with the destination information. A WrittenDestination folder for an MQOutput node has the following format:

```
WrittenDestination = (
   MQ  = (
      DestinationData = (
         queueName        = 'OUT'
         queueManagerName = 'MYQUEUEMANAGER'
         replyIdentifier = X'4d...2e'
         msgId           = X'3c...2c'
         correlId        = X'2a...00'
         GroupId         = X'3a...00'
      )
   )
)
```

Configuring the MQReply node

When you have put an instance of the MQReply node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view.

Configure the MQReply node as follows:

1. Optional: On the **Description** tab, enter a short description, a long description, or both. You can also rename the node on this tab.
2. On the **Advanced** tab:
   a. Select Segmentation Allowed if you want WebSphere MQ to break the message into segments in the queue manager, when appropriate. You must also set MQMF_SEGMENTATION_ALLOWED in the MsgFlags field in the MQMD for segmentation to occur.

      More information about the options to which this property maps is available in the *Application Programming Reference* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online.
   b. Choose the persistence mode that you want for the output message.
      - If you select Automatic (the default), the persistence is as specified in the incoming message.
      - If you select Yes, the message is put persistently.
      - If you select No, the message is put non-persistently.
      - If you select As Defined for Queue, the message persistence is set as defined in the WebSphere MQ queue.
   c. Choose the transaction mode that you want for the output message.
      - If you select Automatic (the default), the message transactionality is derived from how it was specified at the MQInput node.
      - If you select Yes, the message is put transactionally.
      - If you select No, the message is put non-transactionally.
3. On the **Validation** tab, set the validation properties; see "Validation properties" on page 1445. If a message is propagated to the Failure terminal of the node, it is not validated.

   For more details, see "Validating messages" on page 204.

The reply message is put (using MQPUT) to the queue named in the input message MQMD as the ReplyTo queue. You cannot change this destination.

Connecting the output terminals to another node:

Connect the Out or Failure terminal of this node to another node in the message flow to process the message further, process errors, or send the message to an additional destination.

If you use aggregation in your message flows, you must connect these output terminals.

Configuring for coordinated transactions:

When you define an MQReply node, the option that you select for the Transaction Mode property defines whether the message is written under sync point:

- If you select Yes, the message is written under sync point (that is, in a WebSphere MQ unit of work).
- If you select Automatic (the default), the message is written under sync point if the incoming input message is marked as persistent.
- If you select No, the message is not written under sync point.

Another property of the MQReply node, Persistence Mode, defines whether the output message is marked as persistent when it is put to the output queue:

- If you select Yes, the message is marked as persistent.
- If you select Automatic (the default), the message persistence is determined by the properties of the incoming message, as set in the MQMD (the WebSphere MQ message descriptor).
- If you select No, the message is not marked as persistent.
- If you select As Defined for Queue, the message persistence is set as defined in the WebSphere MQ queue; the MQReply node specifies the MQPER_PERSISTENCE_AS_Q_DEF option in the MQMD.

Terminals and properties

The MQReply node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if a failure is detected when the message is put to the output queue.
Out	The output terminal to which the message is routed if it has been successfully put to the output queue, and if further processing is required in this message flow.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory*; the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file for deployment).

The MQReply node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type.	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The MQReply node Advanced properties are described in the following table.

Property	M	C	Default	Description
Segmentation Allowed	Yes	No	Cleared	If you select this check box, WebSphere MQ breaks the message into segments in the queue manager.
Persistence Mode	Yes	No	Automatic	This property controls whether the message is put persistently. Valid values are Automatic, Yes, No, and As Defined for Queue.
Transaction Mode	Yes	No	Automatic	This property controls whether the message is put transactionally. Valid values are Automatic, Yes, and No.

The Validation properties of the MQReply node are described in the following table.

For a full description of these properties, see "Validation properties" on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	Inherit	This property controls whether validation takes place. Valid values are None, Content and Value, Content, and Inherit.	validateMaster
Failure Action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The MQReply node also has the following properties that you cannot access or modify through the workbench interface. However, these values are used by the broker when the message is processed in the message flow.

Property	Description
Queue Manager Name	The name of the WebSphere MQ queue manager to which the output queue, identified in Queue Name, is defined. This name is retrieved from the ReplyTo field of the MQMD header of the input message.
Queue Name	The name of the WebSphere MQ queue to which the output message is put. This name is retrieved from the ReplyTo field of the MQMD header of the input message.
Destination	This property always has the value reply.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Output node

Use the Output node as an out terminal for an embedded message flow (a subflow).

This topic contains the following sections:

Purpose

You can use a subflow for a common task that can be represented by a sequence of message flow nodes. For example, you can create a subflow to increment or decrement a loop counter, or to provide error processing that is common to a number of message flows.

You must use an Output node to provide the Out terminal to a subflow; you cannot use a standard output node (a built-in output node such as MQOutput, or a user-defined output node).

You can include one or more Output nodes in a subflow. Each one that you include provides a terminal through which you can propagate messages to subsequent nodes in the message flow in which you include the subflow.

The Output node is contained in the **Construction** drawer of the palette, and is represented in the workbench by the following icon:

When you select and include a subflow in a message flow, it is represented by the following icon:

When you include the subflow in a message flow, this icon exhibits a terminal for each Output node that you included in the subflow, and the name of the terminal (which you can see when you move your mouse pointer over it) matches the name of that instance of the Output node. Give your Output nodes meaningful names, you can easily recognize them when you use their corresponding terminal on the subflow node in your message flow.

Using this node in a message flow

Look at the following sample to see how to use this node:
- Error Handler

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

Terminals and properties

When you have put an instance of the Output node into a message flow, you can configure it; see “Configuring a message flow node” on page 276. The properties of the node are displayed in the Properties view.

The Output node terminals are described in the following table.

Terminal	Description
In	The output terminal that defines an out terminal for the subflow.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Output node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, Output	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see “Configuring monitoring event sources using monitoring properties” on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Passthrough node

Use the Passthrough node to enable version control of a subflow at run time.

This topic contains the following sections:

Purpose

Use the Passthrough node to add a label to your message flow or subflow. By combining this label with keyword replacement from your version control system, you can identify which version of a subflow is included in a deployed message

flow. You can use this label for your own purposes. If you have included the correct version keywords in the label, you can see the value of the label:

- Stored in the broker archive (BAR) file, using the mqsireadbar command
- As last deployed to a particular broker, on the properties of a deployed message flow in the workbench
- At run time, if you enable user trace for that message flow

The Passthrough node does not process the message in any way. The message that it propagates on its Out terminal is the same message that it received on its In terminal.

The Passthrough node is contained in the **Construction** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Use this node to identify a subflow; for example, if you develop an error processing subflow to include in several message flows, you might want to modify that subflow. However, you might want to introduce the modified version initially to just a subset of the message flows in which it is included. Set a value for the instance of the Passthrough node that identifies which version of the subflow you have included.

Terminals and properties

When you have put an instance of the Passthrough node into a message flow, you can configure it. For more information, see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The Passthrough node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Out	The input terminal that delivers a message to the subflow.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined), the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Passthrough node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	Passthrough	The name of the node.
Short description	No	No		A brief description of the node.

Property	M	C	Default	Description
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Passthrough node Basic properties are described in the following table.

Property	M	C	Default	Description
Label	No	No		The label (identifier) of the node. Enter a value that defines a unique characteristic; for example, the version of the subflow in which the node is included.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

PeopleSoftInput node

Use the PeopleSoftInput node to interact with a PeopleSoft application.

This topic contains the following sections:

Purpose

Use the PeopleSoftInput node to interact with PeopleSoft applications. For example, a PeopleSoftInput node monitors a PeopleSoft system for a specified event. When that event occurs, the PeopleSoftInput node generates a message tree that represents the business object with the new event details. The message tree is propagated to the Out terminal so that the rest of the message flow can use the data to update other systems, or audit the changes.

The PeopleSoftInput node is contained in the **WebSphere Adapters** drawer of the message flow node palette, and is represented in the workbench by the following icon:

Using this node in a message flow

To function correctly, the PeopleSoftInput node needs an adapter component, which you set using the Adapter component node property, and business object definitions, which are stored in the message set that you reference from the node.

For this reason, you must provide a message set. By default, the message that is propagated from the PeopleSoftInput node is in the DataObject domain, so the Message domain property is set to DataObject. You cannot specify a different domain. The message type is detected automatically by the node.

The PeopleSoftInput node populates the route to label destination list with the name of the method binding. If you add a RouteToLabel node to the message flow after the PeopleSoftInput node, the RouteToLabel node can use the name of the method binding to route the message to the correct part of the message flow for processing.

You can deploy only one input node that uses a particular adapter component to an execution group, but you can deploy many input nodes that use different adapter components to an execution group.

You can use the mqsisetdbparms command in the following format to configure an account name with a user name and password for the Adapter for PeopleSoft Enterprise.

```
mqsisetdbparms broker name -n adapter name -u user name -p password
```

For example:

```
mqsisetdbparms BRK1 -n eis::PeopleSoftCustomerInbound.inadapter -u peoplesoftuid -p ********
```

Using configurable services for PeopleSoft nodes

PeopleSoft nodes can get PeopleSoft connection details from either the adapter component or a configurable service. By using a configurable service, you can change the connection details for an adapter without the need to redeploy the adapter. For more details about creating, changing, reporting, and deleting the configurable services for PeopleSoft, see "Changing connection details for PeopleSoft adapters" on page 301.

Terminals and properties

When you have put an instance of the PeopleSoftInput node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. If you double-click a PeopleSoftInput node, you open the Adapter Connection wizard. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The PeopleSoftInput node terminals are described in the following table.

Terminal	Description
Out	Business object events from the adapter are sent to the Out terminal.
Failure	If an error happens in the PeopleSoftInput node, the message is propagated to the Failure terminal. Information about the error, and business object events can also be propagated to the Failure terminal.
Catch	Business object events are sent to the Catch terminal if they cause an uncaught exception in the message flow. If the Catch terminal is not connected, the retry process is activated to handle the business object.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a

value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The PeopleSoftInput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, PeopleSoftInput.	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The PeopleSoftInput node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Adapter component	Yes	Yes		The name of the adapter component that contains configuration properties for the adapter. Either enter a name of an adapter file or click **Browse** to select an adapter file from the list of files that are available in referenced message set projects.	adapterComponent

The PeopleSoftInput node Routing properties are described in the following table.

Property	M	C	Default	Description
Set destination list	No	No	Selected	This property specifies whether to add the method binding name to the route to label destination list. If you select this check box, the method binding name is added so that you can use a RouteToLabel node in the message flow after the PeopleSoftInput node.
Label prefix	No	No		The prefix to add to the method name when routing to label. Add a label prefix to avoid a clash of corresponding label nodes when you include multiple WebSphere Adapters input nodes in the same message flow. By default, there is no label prefix, so the method name and label name are identical.

The PeopleSoftInput node Input Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	DataObject	The domain that is used to parse the incoming message. By default, the message that is propagated from the PeopleSoftInput node is in the DataObject domain. You cannot specify a different domain.
Message set	Yes	No	Set automatically	The name of the message set in which the incoming message is defined. This field is set automatically from the Adapter component property. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Message type	No	No		The name of the incoming message. The node detects the message type automatically. You cannot set this property.

Property	M	C	Default	Description
Message format	No	No		The name of the physical format of the incoming message. You cannot set this property.

The PeopleSoftInput node Transactionality properties are described in the following table.

Property	M	C	Default	Description
Transaction mode	No		Yes	The transaction mode on this input node determines whether the rest of the nodes in the flow operate under sync point.

The Instances properties of the PeopleSoftInput node are described in the following table. For a full description of these properties, refer to "Configurable message flow properties" on page 1324.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Additional instances pool	No	Yes	Use Pool Associated with Message Flow	The pool from which additional instances are obtained. • If you select Use Pool Associated with Message Flow, additional instances are obtained from the message flow value. • If you select Use Pool Associated with Node, additional instances are allocated from the node's additional instances based on the number specified in the Additional instances property.	componentLevel
Additional instances	No	Yes	0	The number of additional instances that the node can start if the Additional instances pool property is set to Use Pool Associated with Node. By default, no additional instances are given to the node.	additionalInstances

The PeopleSoftInput node Retry properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Retry mechanism	No	No	Failure	This property specifies how retry processing is handled when a failure is rolled back to the PeopleSoftInput node. • If you select Failure, retry processing is not performed so you cannot set the remaining properties. • If you select Short and long retry, retry processing is performed first at the interval that is specified by the Short retry interval property, and if that is unsuccessful, it is then performed at the interval that is specified by the Long retry interval property.	
Retry threshold	No	Yes	0	The maximum number of times that retry processing is performed for short retry.	retryThreshold
Short retry interval	No	Yes	0	The interval between short retry attempts.	shortRetryInterval
Long retry interval	No	Yes	0	The interval between long retry attempts.	longRetryInterval

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

PeopleSoftRequest node

Use the PeopleSoftRequest node to interact with a PeopleSoft application.

This topic contains the following sections:

Purpose

Use the PeopleSoftRequest node to interact with PeopleSoft applications. For example, a PeopleSoftRequest node requests information from a PeopleSoft Enterprise Information System (EIS). A customer business object is sent to PeopleSoft, causing PeopleSoft to retrieve information about a customer, such as an address and account details. The response information that is retrieved by the PeopleSoftRequest node can then be used by the rest of the message flow. The PeopleSoftRequest node can send and receive business data.

The PeopleSoftRequest node is contained in the **WebSphere Adapters** drawer of the message flow node palette, and is represented in the workbench by the following icon:

Using this node in a message flow

To function correctly, the PeopleSoftRequest node needs an adapter component, which you set using the Adapter component node property, and business object definitions, which are stored in the message set that you reference from the node. For this reason, you must provide a message set. By default, the message that is propagated from the PeopleSoftRequest node is in the DataObject domain, so the Message domain property is set to DataObject. You cannot specify a different domain. The message type is detected automatically by the node.

The PeopleSoftRequest node supports local transactions using the broker's Local Transaction Manager, and global transactions using the broker's external syncpoint coordinator.

You can deploy several WebSphere Adapters request nodes that use the same adapter component to an execution group.

You can use the mqsisetdbparms command in the following format to configure an account name with a user name and password for the Adapter for PeopleSoft Enterprise.

```
mqsisetdbparms broker name -n adapter name -u user name -p password
```

For example:

```
mqsisetdbparms BRK1 -n eis::PeopleSoftCustomerOutbound.outadapter -u peoplesoftuid -p ********
```

Using configurable services for PeopleSoft nodes

PeopleSoft nodes can get PeopleSoft connection details from either the adapter component or a configurable service. By using a configurable service, you can change the connection details for an adapter without the need to redeploy the adapter. For more details about creating, changing, reporting, and deleting the configurable services for PeopleSoft, see "Changing connection details for PeopleSoft adapters" on page 301.

Terminals and properties

When you have put an instance of the PeopleSoftRequest node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. If you double-click a PeopleSoftRequest node, you open the Adapter Connection wizard. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The PeopleSoftRequest node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts the request business object.
Out	The output terminal to which the response business object is sent if it represents successful completion of the request, and if further processing is required within this message flow.
Failure	If an error happens in the PeopleSoftRequest node, the message is propagated to the Failure terminal. Information about the error, and business object events can also be propagated to the Failure terminal.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk on the panel if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The PeopleSoftRequest node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, PeopleSoftRequest	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The PeopleSoftRequest node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Adapter component	Yes	No		The name of the adapter component that contains configuration properties for the adapter. Either enter a name of an adapter file, or click **Browse** to select an adapter file from the list of files that are available in referenced message set projects.	
Default method	Yes	Yes		The default method binding to use.	defaultMethod

The PeopleSoftRequest node Response Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	DataObject	The domain that is used to parse the response message. By default, the response message that is propagated from the PeopleSoftRequest node is in the DataObject domain. You cannot specify a different domain.
Message set	Yes	No	Set automatically	The name of the message set in which the response message is defined. This field is set automatically from the Adapter component property. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Message type	No	No		The name of the response message. The node detects the message type automatically. You cannot set this property.
Message format	No	No		The name of the physical format of the response message. You cannot set this property.

The PeopleSoftRequest node Transactionality properties are described in the following table.

Property	M	C	Default	Description
Transaction mode	No	No	No	This property specifies that updates are performed independently, not as part of a local transaction. You cannot change this property.

The PeopleSoftRequest node Request properties are described in the following table.

Property	M	C	Default	Description
Method Location	Yes	No	$LocalEnvironment/Adapter/MethodName	The location of the business method (such as createPurchaseOrder or deletePurchaseOrder) that is used to trigger the PeopleSoftRequest node to perform an action on the external system.
Data Location	Yes	No	$Body	The location in the incoming message tree from which data is retrieved to form the request that is sent from the PeopleSoftRequest node to the EIS.

The PeopleSoftRequest node Result properties are described in the following table.

Property	M	C	Default	Description
Output data location	No	No	$OutputRoot	The message tree location to which the PeopleSoftRequest node sends output.
Copy local environment	No	No	Selected	This property controls how the local environment is copied to the output message. If you select the check box, at each node in the message flow, a new copy of the local environment is created in the tree, and it is populated with the contents of the local environment from the preceding node. So if a node changes the local environment, the upstream nodes do not see those changes because they have their own copies. This behavior might be an issue if you are using a FlowOrder node, or if you use the propagate command on a Compute node. If you clear the check box, each node does not generate its own copy of the local environment, but it uses the local environment that is passed to it by the previous node. So if a node changes the local environment, those changes are seen by the upstream nodes.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

PHPCompute node

Use the PHPCompute node to route and transform an incoming message, using the PHP scripting language.

The PHPCompute node is available on Windows only.

This topic contains the following sections:

Purpose

The PHPCompute node can use the PHP scripting language to route and transform incoming messages.

Using this node, you can achieve the following goals:
- Examine an incoming message and, depending on its content, propagate it unchanged to the node's output terminal.
- Change part of an incoming message and propagate the changed message to the output terminal by using PHP.

- Create and build a new output message that is independent of the input message by using PHP.

The PHPCompute node is contained in the **Transformation** drawer of the palette, and is represented in the workbench by the following icon:

To enable function that becomes available in WebSphere Message Broker fix packs, use the **-f** parameter on the mqsichangebroker command. For more information, see mqsichangebroker command.

Using the PHPCompute node in a message flow

Look at the following sample to see how to use this node:

- PHPCompute Node

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

Specifying PHP

Create PHP statements to customize the behavior of the PHPCompute node. For example, you can customize the node to create one or more output messages by using either new data or the content of an input message or database (unchanged or modified). For example, you might want to modify a value in the input message by adding a value from a database, and store the result in a field in the output message.

Create the PHP statements that you want in a PHP script file and ensure that it exists in the workspace before you associate it with the PHPCompute node.

If the required PHP script file exists, import it into the workspace before associating it with the PHPCompute node (see Importing file systems into the workbench).

If a PHP file does not exist for this node, create one in the project folder with a file extension of `.php` (for example, `myfile.php`). For more information about creating a PHP script file, see “Creating PHP code for a PHPCompute node” on page 456.

Configuring the PHPCompute node

When you have put an instance of the PHPCompute node into a message flow, you can configure it. For more information about how to configure nodes, see “Configuring a message flow node” on page 276.

The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (the ones that do not have a default value defined) are marked with an asterisk in that view.

To configure the PHPCompute node:

1. Optional: On the **Description** tab, enter a short description, a long description, or both. You can also rename the node on this tab.

2. On the **Basic** tab, use the PHP script property to specify the name of the PHP file. Select the Invoke 'evaluate()' method in PHP class definition property if the code in your PHP script file includes an evaluate method.
3. On the **Parser options** tab, select the Use XMLNSC compact parser for the XMLNS domain property to specify that the XMLNSC Compact Parser is used for messages in the XMLNS Domain.
4. On the **Validation** tab, specify the parser validation properties of the node. For more information about validation, see "Validating messages" on page 204. For information about how to complete this tab, see "Validation tab properties" on page 1446.

Terminals and properties

The PHPCompute node terminals are described in the following table.

Terminal	Type	Description
In	Input data	The input terminal that accepts a message for processing by the node.
Out	Output data	The output terminal to which the transformed message is routed.
Failure	Output data	The output terminal to which the message is routed if a failure is detected during the computation. Even if the Validate property is set, messages that are propagated to the Failure terminal of the node are not validated.
* (dynamic)	Dynamic output	Zero or more dynamic output terminals can be created to support message routing.

You can define additional dynamic output terminals on the PHPCompute node. Not all dynamic output terminals that are created on a PHPCompute node need to be mapped to an expression in the filter table. If any of the dynamic output terminals are unmapped, they never have messages propagated to them. Several expressions can map to the same single dynamic output terminal. No static output terminal exists to which the message is passed straight through. For more information about using dynamic terminals, see "Using dynamic terminals" on page 279.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the PHPCompute node are described in the following table:

Property	M	C	Default	Description
Node name	No	No	PHPCompute	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the PHPCompute node are described in the following table:

Property	M	C	Default	Description	mqsiapplybaroverride command property
PHP script	Yes	Yes		A string containing the name of the PHP script file.	ScriptName

The Parser Options properties of the PHPCompute node are described in the following table.

Property	M	C	Default	Description
Use XMLNSC compact parser for XMLNS domain	No	No	False	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the message data appears under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or Input Message Parsing property, Message Domain, is XMLNS.

The Validation properties of the PHPCompute node are described in the following table.

For a full description of these properties, see "Validation properties" on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	No	None	This property controls whether validation takes place. Valid values are: • None • Content and Value • Content • Inherit	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are: • User Trace • Local Error Log • Exception • Exception List	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Publication node

Use the Publication node to filter output messages from a message flow and transmit them to subscribers who have registered an interest in a particular set of topics. The Publication node must always be an output node of a message flow and has no output terminals of its own.

This information contains the following sections:
- "Purpose"
- "Using this node in a message flow"
- "Terminals and properties" on page 1127

Purpose

Use the Publication node if your message flow supports publish/subscribe applications. Applications that expect to receive publications must register a subscription with a broker, and can optionally qualify the publications that they get by providing restrictive criteria (such as a specific publication topic).

If your subscriber applications use the WebSphere MQ Enterprise Transport to connect to the broker, you can define the queues to which messages are published as WebSphere MQ clustered queues or shared queues.

Publications can also be sent to subscribers within a WebSphere MQ cluster if a cluster queue is nominated as the subscriber queue. In this case, the subscriber should use the name of an "imaginary" queue manager that is associated with the cluster, and ensure that a corresponding blank queue manager alias definition for this queue manager is made on the broker that satisfies the subscription.

The Publication node is contained in the **Routing** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Look at the following samples to see how to use this node:
- Soccer Results
- Scribble
- JMS Nodes
- Pager

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

For an example of how to use this node, assume that you have written a publishing application that publishes stock updates on a regular basis. The application sends the messages to the broker on an MQInput node, and the message flow provides a conversion from the input currency to a number of output currencies. Include a Publication node for each currency that is supported, and set the Subscription Point property to a value that reflects the currency in which the stock price is published by the node; for example, Sterling, or USD.

Terminals and properties

When you have put an instance of the Publication node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view.

The Publication node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory*; the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Publication node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type: Publication	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Publication node Basic properties are described in the following table.

Property	M	C	Default	Description
Implicit Stream Naming	Yes	No	Cleared	Select Implicit Stream Naming to take the name of the WebSphere MQ queue on which the message was received by the message flow as the stream name. This property provides compatibility with earlier versions of WebSphere MQ Publish/Subscribe, and applies to messages with an MQRFH header when MQPSStream is not specified. Clear the check box if you do not want this action to be taken.
Subscription Point	No	No		The subscription point value for the node. If you do not specify a value for this property, the default subscription point is assumed. Set a subscription point for a Publication node to restrict the forwarding of its publications to those subscribers that specify the subscription point in their subscription (as described in the example scenario in "Using this node in a message flow" on page 1126). For more information, see Subscription points.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Real-timeInput node

Use the Real-timeInput node to receive messages from clients that connect to the broker using the WebSphere MQ Real-time Transport or the WebSphere MQ Multicast Transport, and that use JMS application programming interfaces.

This topic contains the following sections:

Purpose

The Real-timeInput node handles messages in the following message domains:
- JMSMap
- JMSStream

An output node in a message flow that starts with a Real-timeInput node can be any of the supported output nodes, including user-defined output nodes. You can create a message flow that receives messages from real-time clients and generates messages for clients that use all supported transports to connect to the broker, because you can configure the message flow to request the broker to provide any conversion that is required.

If you create a message flow to use as a subflow, you cannot use a standard input node; you must use an instance of the Input node as the first node to create an In terminal for the subflow.

If your message flow does not receive messages from JMS applications, choose one of the supported input nodes.

The Real-timeInput node is contained in the **Additional Protocols** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Look at the following sample to see how you can use this node:
- Scribble

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

Terminals and properties

When you have put an instance of the Real-timeInput node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The Real-timeInput node terminals are described in the following table.

Terminal	Description
Out	The output terminal to which the message is routed if it is successfully retrieved from JMS. If this routing fails, the message is tried again.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined), the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Real-timeInput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type: Real-timeInput	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Real-timeInput node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Port	Yes	Yes		The port number on which the input node listens for publish or subscribe requests from JMS applications. Ensure that the port number that you specify does not conflict with any other listener service. No default value is provided for this property; you must enter a value.	port
Authentication	Yes	No	Cleared	To authenticate users that send messages on receipt of their messages, select this check box. If you clear the check box (the default setting), users are not authenticated.	
Tunnel through HTTP	Yes	No	Cleared	Select the check box to indicate that users use HTTP tunneling. If you clear the check box (the default setting), messages do not use HTTP tunneling. If you select the check box, all client applications that connect must use this feature. If they do not use this feature, their connection is rejected. The client application cannot use this option in conjunction with the connect-via proxy setting, which is activated from the client side.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Read Threads	No	Yes	10	The number of threads that you want the broker to allocate to read messages. The broker starts as many instances of the message flow as are necessary to process current messages, up to this limit.	readThreads
Write Threads	No	Yes	10	The number of threads that you want the broker to allocate to write messages. The broker starts as many instances of the message flow as are necessary to process current messages, up to this limit.	writeThreads
Authentication Threads	No	Yes	10	The number of threads that you want the broker to allocate to user authentication checks. The user authentication check is performed when a message is received. The broker starts as many instances of the message flow as are necessary to process current messages, up to this limit.	authenticationThreads

The properties of the General Message Options for the Real-timeInput node are described in the following table.

Property	M	C	Default	Description
Parse Timing	No	No	On Demand	This property controls when an input message is parsed. Valid values are On Demand, Immediate, and Complete. Parse timing is, by default, set to On Demand, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see "Parsing on demand" on page 1449.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Real-timeOptimizedFlow node

Use the Real-timeOptimizedFlow node to receive messages from clients that connect to the broker using the WebSphere MQ Real-time Transport or the WebSphere MQ Multicast Transport, and that use JMS application programming interfaces.

This topic contains the following sections:

Purpose

The Real-timeOptimizedFlow node is a complete message flow that provides a high performance publish/subscribe message flow. The actions that are taken by

this node are all internalized; you cannot influence the node's operation except by configuring its properties, and you cannot connect it to any other node.

This node also supports publication to, or subscription from, standard WebSphere MQ applications, but its performance for these applications is not as good as the performance achieved for JMS applications.

You cannot affect the message content in any way when you use the Real-timeOptimizedFlow node. To modify the input message, or to send messages or make publications available to applications that use other communications protocols, use the Real-timeInput node.

Include the Real-timeOptimizedFlow node in a message flow when you want to distribute messages through a broker to and from client applications that use JMS.

The Real-timeOptimizedFlow node is contained in the **Additional Protocols** drawer of the palette, and is represented in the workbench by the following icon:

You cannot include this node in a message flow if you have disabled the publish/subscribe engine of the broker or brokers to which you intend to deploy it. If you deploy a BAR file that includes this message flow, the deployment succeeds, but the broker throws a recoverable exception when a message is received by this node.

Terminals and properties

When you have put an instance of the Real-timeOptimizedFlow node into a message flow, you can configure it. For more information, see “Configuring a message flow node” on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The Real-timeOptimizedFlow node has no terminals. It is a complete message flow and cannot be connected to other nodes to extend the message processing.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined), the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Real-timeOptimizedFlow node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	Real-timeOptimizedFlow	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Real-timeOptimizedFlow node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Port	Yes	Yes		The port number on which the node listens for publish or subscribe requests from JMS applications. Ensure that the port number that you specify does not conflict with any other listener service. No default value is provided for this property; you must enter a value.	port
Authentication	Yes	No	Cleared	For users to authenticate that send messages on receipt of their messages, select Authentication. If you clear the check box (the default setting), users are not authenticated.	
Tunnel through HTTP	Yes	No	Cleared	For clients to use HTTP tunneling, select Tunnel through HTTP. If you clear the check box (the default setting), messages do not use HTTP tunneling. If you select the check box, all client applications that connect must use this feature. If they do not use this feature, their connection is rejected. The client application cannot use this option in conjunction with the connect-via-proxy setting, which is activated from the client side.	
Read threads	No	Yes	10	The number of threads that you want the broker to allocate to read messages. The broker starts as many instances of the message flow as are necessary to process current messages, up to this limit.	readThreads
Write threads	No	Yes	10	The number of threads that you want the broker to allocate to write messages. The broker starts as many instances of the message flow as are necessary to process current messages, up to this limit.	writeThreads
Authentication threads	No	Yes	10	The number of threads that you want the broker to allocate to user authentication checks. The user authentication check is performed when a message is received. The broker starts as many instances of the message flow as are necessary to process current messages, up to this limit.	authenticationThreads

RegistryLookup node

Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. The input message is not modified.

This topic contains the following sections:

Purpose

The RegistryLookup node retrieves any type of entity held in WSRR.

Data is retrieved according to search criteria defined by node properties, possibly supplemented or overridden by local environment definitions at run time; see "Local environment overrides" on page 1134 for more details.

The retrieved data is placed in the local environment tree, making it available to subsequent nodes. The input message received by the node is propagated to the output terminal unchanged.

RegistryLookup node processing

The RegistryLookup node is contained in the **Web services** drawer of the message flow node palette, and is represented in the workbench by the following icon:

When the RegistryLookup node receives a message the following steps occur in sequence.

1. The RegistryLookup node retrieves the data from WSRR using the specified search criteria.
2. If one or more matches are found, the RegistryLookup node adds a representation of those entities to the local environment tree.
   - If Match Policy is set to One, a single entity is returned by WSRR and added to the local environment tree. If the registry contains more than one entity that matches the specified search criteria it is not possible to determine which one is returned by WSRR. A different one can be returned each time the query is issued.
   - If Match Policy is set to All, all matching entities are added to the local environment tree. The order of the entities is determined by WSRR and might vary between queries.

   The input message is propagated unchanged to the Output terminal. The local environment tree is propagated to the Out terminal, where it is available for further processing by transformation nodes. The exact representation of an entity in the local environment tree depends on the Depth Policy property. See "RegistryLookup node output" on page 795 for details of the local environment output tree.

   Add a compute node to your message flow to interpret and act on the returned data. For instance, if the local environment tree contains multiple endpoint addresses for use by subsequent request nodes the Compute node should select the required address and set up any transport headers or local environment settings required by those request nodes.
3. If no matches are found, the RegistryLookup node propagates the input message to the NoMatch terminal.
4. If a processing error occurs, for example if the WSRR server configured on the `DefaultWSRR` configurable service object cannot be connected to, or the connection times out, the RegistryLookup node propagates the input message unchanged to the Failure terminal. The ExceptionList is populated with details of the error.

Local environment overrides

You can override the RegistryLookup node properties by using local environment settings. See “Dynamically defining the search criteria” on page 791.

Terminals and properties

When you have put an instance of the RegistryLookup node into a message flow, you can configure it; see “Configuring a message flow node” on page 276. The properties of the node are displayed in the Properties view.

The RegistryLookup node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if an error occurs within the node's processing.
Out	The output terminal to which the unmodified input message and updated local environment containing the matched registry data is sent.
NoMatch	The terminal to which the input message is sent if no matching entity is found based on the specified search criteria.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The RegistryLookup node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type (RegistryLookup)	The name of the node
Short description	No	No	None	A brief description of the node
Long description	No	No	None	Text that describes the purpose of the node in the message flow

The RegistryLookup node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Name	No	Yes	None	Enter the string values for one or more of Name, Namespace, and Version for the entities or artifacts that you want to retrieve from WSRR. At least one of the properties is required. If you leave all three property values blank, you will get an error message when you try to save.	name
Namespace	No	Yes	None		namespace
Version	No	Yes	None		serviceVersion
Template	No	Yes	None	The Template property is deprecated; use a user-defined property called *template*.	template

Property	M	C	Default	Description	mqsiapplybaroverride command property
User Properties	No	No	None	Allows a query to specify user-defined properties. Add User Properties by clicking **Add**. User Properties refer to the Additional Properties that are used to catalogue the entities in WSRR. Enter values for Property Name, which is the case sensitive match of the additional property in WSRR, Property Type, and Property Value. The Property Type can be: • a String (the default), in which case the Property Value is a character string to be matched with the additional property value present in WSRR • XPATH, or ESQL, in which case the Property Value is a XPath or ESQL expression which locates a field in the message tree that contains the character string to be matched with the property value present in WSRR.	
Classification	No	No	None	The Web Ontology Language (OWL) classification system property. Each classifier is a class in OWL, and has a Uniform Resource Identifier (URI). Using classifications in the registry can help to make objects easier to find and can also add meaning to custom objects that are unique to a particular system. Add a Classification by clicking **Add** and typing the complete fully qualified OWL URI for the OWL classification. For example, it can define a particular service endpoint's lifecycle state.	
Match Policy	Yes	No	One	WSRR can contain multiple entities that match the search criteria specified by the properties above. If Match Policy is set to One, at most one matching entity is returned. If Match Policy is set to All, all matching entities are returned. See “RegistryLookup node output” on page 795	

The RegistryLookup node Advanced properties are described in the following table.

Property	M	C	Default	Description
Depth Policy	Yes	No	Return matched showing immediate relationships (For compatibility only)	Specify the depth of the WSRR query and the contents of the entity data to be returned. The returned entities are stored in the ServiceRegistry message tree in the LocalEnvironment. Select Return matched only (Depth = 0) to use a WSRR query depth of 0, and to return only the matched entities. Select Return matched showing immediate relationships (For compatibility only) to use a WSRR query depth of 1, and to return only the matched entities. Returned entities include elements listing the bsrURIs of related child entities. This option provides compatibility with versions of WebSphere Message Broker before Version 6.1.0.4, using the output format that was used in those prior versions. This option is deprecated, use one of the other options. Select Return matched plus immediate related entities (Depth = 1) to use a WSRR query depth of 1, and to return the matched entities and the immediate related child entities. Select Return matched plus all related entities (Depth = -1) to use a WSRR query depth of -1, and to return the matched entities and all the related child entities.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

ResetContentDescriptor node

Use the ResetContentDescriptor node to request that the message is reparsed by a different parser.

This topic contains the following sections:

Purpose

If you specify MRM as the new parser, you can also specify a different message template (message set, message type, and message format). This node does not reparse the message, but the properties that you set for this node determine how the message is parsed when it is next reparsed by the message flow.

The node associates the new parser information with the input message bit stream. If the message has been parsed already to create a message tree, and the contents

of the tree have been modified (for example, by a Compute node), the ResetContentDescriptor node must re-create the bit stream from the message tree by calling the current parser.

If your message flow has updated the message before it is received by the ResetContentDescriptor node, ensure that the changed message contents are still valid for the current parser. If the contents are not valid, the parser generates an error when it attempts to re-create the bit stream from the message tree, and the ResetContentDescriptor node raises an exception. For example, if you have added a new field to a message in the MRM domain, and the field is not present in the model, the re-creation of the bit stream fails.

The ResetContentDescriptor node does not:

- Change the message content; it changes message properties to specify the way in which the bit stream is parsed next time that the parser is started.
- Convert the message from one format to another; for example, if the incoming message has a message format of XML and the outgoing message format is binary, the ResetContentDescriptor node does not do any reformatting. It starts the parser to re-create the bit stream of the incoming XML message, which retains the XML tags in the message. When the message is reparsed by a subsequent node, the XML tags are not valid and the reparse fails.

The ResetContentDescriptor node is contained in the **Construction** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

For an example of how to use this node, assume that you want to swap between the BLOB and the MRM domains. The format of an incoming message might be unknown when it enters a message flow, therefore the BLOB parser is started. Later on in the message flow, you might decide that the message is predefined as a message in the MRM domain, and you can use the ResetContentDescriptor node to set the correct values to use when the message is parsed by a subsequent node in the message flow.

The following table shows typical ResetContentDescriptor node properties.

Property	Value
Message domain	MRM
Reset message domain	Selected
Message set	MyMessageSet
Reset message set	Selected
Message type	m_MESSAGE1
Reset message type	Selected
Message format	Text1
Reset message format	Selected

The Message domain is set to MRM, and the MRM parser is started when the message is next parsed. The Message set, Message type, and Message format are the message template values that define the message model, and all of the reset check boxes are selected because all of the properties need to change.

The ResetContentDescriptor node causes the BLOB parser that is associated with the input message to construct the physical bit stream of the message (not the logical tree representation of it), which is later passed to the MRM parser. The MRM parser then parses the bit stream using the message template (Message set, Message type, and Message format) specified in this ResetContentDescriptor node.

In Version 6.1, you do not have to include a ResetContentDescriptor node after an XSLTransform node in your message flow to set Message domain, Message set, Message type, and Message format of the message generated by the XSLTransform node. The XSLTransform node performs this function.

Configuring the ResetContentDescriptor node

When you have put an instance of the ResetContentDescriptor node into a message flow, you can configure the node. For more information, see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view.

All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

1. Optional: On the **Description** tab, enter a Short description, a Long description, or both. You can also rename the node on this tab.
2. On the **Basic** tab:
   a. To use a different parser associated with the message, specify the new domain in the Message domain property:
      - MRM
      - XMLNSC
      - DataObject
      - XMLNS
      - JMSMap
      - JMSStream
      - MIME
      - BLOB
      - XML (this domain is deprecated; use XMLNSC)
      - IDOC (this domain is deprecated; use MRM)

      You can also specify a user-defined parser if appropriate.

      You must also select the Reset message domain check box.

      If you leave the Message domain property blank and do not select the Reset message domain check box, the domain is not reset. If you leave the Message domain property blank and select the Reset message domain check box, the default value is BLOB.

   b. If the MRM, XMLNSC, or IDOC parser is to reparse the message, specify the other properties of the model that are to be associated with the input message, and select the relevant reset check box beneath each field. If you select a reset check box for a property and you have not specified a value for that property, the value of that property is reset to blank. Alternatively, if you have specified a value for that property, the property is not blank. If you do not select the reset check box for a property, the value for that

property is inherited from the incoming message. If the parser is associated with the input message already, specify only the properties that are to change.

1) Define the Message set. Choose a value from the list of available message sets (the name and identifier of the message set are shown), and select the Reset message set check box.
2) For MRM only, define the name of the message in Message type. Enter the name and select the Reset message type check box.
3) For MRM and IDOC, define the Message format. This property specifies the physical format for the parser. You can select one of the formats from the list (which lists the names of those formats that you have defined on the Message set specified previously), and select Reset message format.

These properties set the message domain, message set, message type, and message format values in the message header of the message that you want to pass through the ResetContentDescriptor node. However, these actions are taken only if suitable headers already exist. If the message does not have an MQRFH2 header, the node does not create one.

3. On the **Parser Options** subtab:
   a. Parse timing is, by default, set to On Demand, which causes parsing of the message to be delayed.

      For more details, see "Parsing on demand" on page 1449.
   b. Select Use MQRFH2C compact parser for MQRFH2 header if you want the MQRFH2C parser to be used. By default, this check box is cleared, which means that the compact parser is not used.
   c. If you are using the XMLNSC parser, set values for the properties that determine how the XMLNSC parser operates. For more information, see "Manipulating messages in the XMLNSC domain" on page 408.
4. On the **Validation** tab, set the validation properties if you want the parser to validate the body of messages against the Message set. (If a message is propagated to the Failure terminal of the node, it is not validated.)

   For more details, see "Validating messages" on page 204 and "Validation properties" on page 1445.

Terminals and properties

The ResetContentDescriptor node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if an error is detected by the node.
Out	The output terminal to which the message is routed if a new parser is identified by the properties.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the ResetContentDescriptor node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the ResetContentDescriptor node are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	BLOB	The message domain that is associated with the message that you want to reparse.
Reset message domain	Yes	No	Cleared	If you select the reset check box, the Message domain property is reset. In this case, if you do not select a value for the Message domain property, the Message domain property value is BLOB.
Message set	No	No		The message set that is associated with the message that you want to reparse. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Reset message set	Yes	No	Cleared	If you select the reset check box, the Message set property is reset. In this case, if you do not select a value for the Message set property, the Message set property value is blank.
Message type	No	No		The message type that is associated with the message that you want to reparse.
Reset message type	Yes	No	Cleared	If you select the reset check box, the Message type property is reset. In this case, if you do not select a value for the Message type property, the Message type property value is blank.
Message format	No	No		The message format that is associated with the message that you want to reparse. If you set the Message domain property to DataObject, you can set this property to XML or SAP ALE IDoc. Set this property to SAP ALE IDoc when you need to parse a bit stream from an external source and generate a message tree.
Reset message format	Yes	No	Cleared	If you select the reset check box, the Message format property is reset. In this case, if you do not select a value for the Message format property, the Message format property value is blank.

The Parser Options properties of the ResetContentDescriptor node are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	On Demand	This property controls when the reparsed message is parsed. Valid values are On Demand, Immediate, and Complete. For a full description of this property, see "Parsing on demand" on page 1449.

Property	M	C	Default	Description
Use MQRFH2C compact parser for MQRFH2 header	No	No	Cleared	This property controls whether the MQRFH2C compact parser, instead of the MQRFH2 parser, is used for MQRFH2 headers.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the XMLNSC parser creates syntax elements in the message tree with data types taken from the XML Schema. You can select this property only if you set the Validate property on the **Validation** tab to Content or Content and Value.
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the message data appears under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or Domain is XMLNS.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in the reparsed message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in the reparsed message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in the reparsed message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the reparsed message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if the value of the Validate property is set to None); entries that are specified in Opaque Elements are ignored if validation is enabled.

The Validation properties of the ResetContentDescriptor node are described in the following table. For a full description of these properties, see "Validation properties" on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are None, Content, Content and Value, and Inherit.	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content and Value or Content. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Route node

Use the Route node to direct messages that meet certain criteria down different paths of a message flow.

This topic contains the following sections:

Purpose

As an example, you can forward a message to different service providers, based on the request details. You can also use the Route node to bypass unnecessary steps. For example, you can check to see if certain data is in a message, and perform a database lookup operation only if the data is missing. If you set the Distribution Mode property to All, you can trigger multiple events that each require different conditions. For example, you can log requests that relate to a particular account identifier, and send requests that relate to a particular product to be audited.

You use XPath filter expressions to control processing. A result of a filter expression is cast as Boolean, so the result is guaranteed to be either true or false. For more information about XPath 1.0 query syntax, see W3C XPath 1.0 Specification.

The Route node is contained in the **Routing** drawer of the message flow node palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Look at the following sample to see how to use this node:

- Simplified Database Routing

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

The Route node has one input terminal and a minimum of three output terminals: Match, Default, and Failure. The Default and Failure output terminals are static, so they are always present on the node. The dynamic Match terminal is created automatically each time a new Route node is selected and used in the Message Flow editor. This behavior means that you do not always have to create the first

dynamic output terminal for this node, which is the minimum number of terminals needed for this node to operate. You can rename this dynamic terminal if "Match" is not an appropriate name.

A message is copied to the Default terminal if none of the filter expressions are true. If an exception occurs during filtering, the message is propagated to the Failure terminal. The Route node can define one or more dynamic output terminals. For all terminals, the associated filter expression is applied to the input message and, if the result is true, a copy of the message is routed to the specified terminal. The Route node determines the order in which the terminals are driven. The node always propagates messages to the terminals in the order in which they appear in the filter table.

Each filter expression is applied to the input message in the order that is given in the filter table. If the result is true, a copy of the message is routed to its associated dynamic output terminal. If you set the Distribution Mode property to First, application of all filter expressions might not occur.

Consider the following example input message:

```
<EmployeeRecord>
 <EmployeeNumber>00001</EmployeeNumber>
 <FamilyName>Smith</FamilyName>
 <Wage>20000</Wage>
</EmployeeRecord>
```

and the following XPath filter expressions:

```
$Body/EmployeeRecord/EmployeeNumber="00002"|Match
$Body/EmployeeRecord/EmployeeNumber="00001"|out_exp2
```

In this example, the Distribution Mode property is set to First. The Route node processes the XPath filter expressions, in the order in which the are displayed, against the input message. Because the Distribution Mode property is set to First, the unmodified input message is propagated only once to the dynamic output terminal that is mapped to the first filter expression that resolves to true. In the previous example, the first filter expression, which is associated with the Match terminal, is false because the employee number in the input message is not "00002". Therefore no message is propagated to the Match terminal. The second filter expression is true, therefore a copy of the input message is routed to the "out_expr2" dynamic terminal. If the employee number in the input message is "00003", and therefore does not match either filter expression, the message is propagated to the static Default output terminal. If the Distribution Mode property is set to All for this example, the same outcome is achieved because only one filter expression is true.

Terminals

The Route node terminals are described in the following table.

Terminal	Description
In	The static input terminal that accepts a message for processing by the node.
Match	A dynamic output terminal to which the original message can be routed when processing completes successfully. You can create additional dynamic terminals; see “Dynamic terminals” on page 1144.
Default	The static output terminal to which the message is routed if no filter expression resolves to true.

Terminal	Description
Failure	The static output terminal to which the message is routed if a failure is detected during processing.

Dynamic terminals

The Route node can have further dynamic output terminals. Not all dynamic output terminals that are created on a Route node have to be mapped to an expression in the filter table. Messages are never propagated to unmapped dynamic output terminals. Several expressions can map to the same single dynamic output terminal. No static output terminal exists to which the message is passed straight through. For more information about using dynamic terminals, see "Using dynamic terminals" on page 279.

Properties

When you have put an instance of the Route node into a message flow, you can configure it. For more information, see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those properties that do not have a default value defined) are marked with an asterisk.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Route node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, Route	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Route node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Filter table	Yes	No		A table where all rows are expressions and associated terminal names that define the switching that is performed by this node following evaluation of each filter expression. The full expression is in the format *XPath filter expression*, *terminal name* All XPath expressions must start with $Root, $Body, $Properties, $LocalEnvironment, $DestinationList, $ExceptionList, or $Environment. Expressions are evaluated in the order in which they are displayed in the table. To improve performance, specify the expressions that are satisfied most frequently at the top of the filter table. Typically, you specify a unique terminal name for each XPath expression.	
Distribution Mode	No	Yes	All	This property determines the routing behavior of the node when an inbound message matches multiple filter expressions. If you set the Distribution Mode property to First, the message is propagated to the associated output terminal of the first expression in the table that resolves to true. If you set this property to All, the message is propagated to the associated output terminal for each expression in the table that resolves to true. If no output terminal matches, the message is propagated to the Default terminal.	distributionMode

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

RouteToLabel node

Use the RouteToLabel node in combination with one or more Label nodes to dynamically determine the route that a message takes through the message flow, based on its content.

This topic contains the following sections:

Purpose

The RouteToLabel node interrogates the local environment of the message to determine the identifier of the Label node to which to route the message.

You must precede the RouteToLabel node in the message flow with a Compute node that populates the local environment of the message with the identifiers of one or more Label nodes that introduce the next sequence of processing for the message. The destinations are set up as a list of label names in the local environment tree in a specific location. This excerpt of ESQL from the Airline Reservations sample demonstrates how to set up the local environment content in a Compute node:

```
IF InputRoot.XMLNSC.PassengerQuery.ReservationNumber<>'' THEN
   SET OutputLocalEnvironment.Destination.RouterList.DestinationData[1].labelName = 'SinglePassenger';
ELSE
   SET OutputLocalEnvironment.Destination.RouterList.DestinationData[1].labelName = 'AllReservations';
END IF;
```

The label names can be any string value, and can be specified explicitly in the Compute node, taken or cast from any field in the message, or retrieved from a database. A label name in the local environment must match the Label Name property of a corresponding Label node.

When you configure the Compute node, you must also select a value for the Compute Mode property from the list that includes LocalEnvironment.

Design your message flow so that a RouteToLabel node logically precedes one or more Label nodes in a message flow, but do not physically wire the RouteToLabel node with a Label node. The connection is made by the broker, when required, according to the contents of the local environment.

The RouteToLabel node is contained in the **Routing** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Look at the following sample to see how to use this node:

- Airline Reservations

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

Terminals and properties

When you have put an instance of the RouteToLabel node into a message flow, you can configure it; see “Configuring a message flow node” on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value are marked with an asterisk.

The RouteToLabel node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if a failure is detected during processing.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The RouteToLabel node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type: RouteToLabel	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The RouteToLabel node Basic properties are described in the following table.

Property	M	C	Default	Description
Mode	Yes	No	Route To Last	This property controls how the RouteToLabel node processes the items in the local environment that is associated with the current message . Valid values are: • Route To First: removes the first item from the local environment. The current message is routed to the Label node that is identified by labelName in that list item. • Route To Last (the default): removes the last item from the local environment. The current message is routed to the Label node that is identified by labelName in that list item.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

SAPInput node

Use the SAPInput node to accept input from an SAP application.

This topic contains the following sections:

Purpose

Use the SAPInput node to accept input from SAP applications. For example, the SAPInput node might monitor an SAP system for new purchase orders. When a new purchase order is raised, the SAPInput node generates a message tree that

represents the business object with the new purchase order details. The message tree is propagated to the Out terminal so that the rest of the message flow can use the data to update other systems, or to audit the changes.

The SAPInput node is contained in the **WebSphere Adapters** drawer of the message flow node palette, and is represented in the workbench by the following icon:

Using this node in a message flow

The SAPInput node needs an adapter component to function correctly. You set the component by using the Adapter component node property and business object definitions, which are stored in the message set that you reference from the node. For this reason, you must provide a message set. By default, the message that is propagated from the SAPInput node is in the DataObject domain, so the Message domain property is set to DataObject. You cannot specify a different domain. The message type is detected automatically by the node.

The SAPInput node populates the route-to-label destination list with the name of the method binding. If you add a RouteToLabel node to the message flow after the SAPInput node, the RouteToLabel node can use the name of the method binding to route the message to the correct part of the message flow for processing.

You can deploy only one input node that uses a particular adapter component to an execution group, but you can deploy many input nodes that use different adapter components to an execution group.

You can use the mqsisetdbparms command in the following format to configure an account name with a user name and password for the Adapter for SAP Software.

```
mqsisetdbparms broker name -n adapter name -u user name -p password
```

For example:

```
mqsisetdbparms BRK1 -n eis::SAPCustomerInbound.inadapter -u sapuid -p ********
```

The SAP inbound adapter has a property called Number of listeners, which configures the adapter to have a particular number of threads listening to the SAP RFC program ID. These threads are not used directly to process messages in a message flow. When a message listener has an event to deliver to the message flow, it requests one of the instances of the flow. In general, it is sensible to keep the number of listeners equal to the number of instances (where instances equals 1 plus additional instances set on the flow or node). For example:

- If the number of listeners is 1, and additional instances is 0, you get a single-threaded message flow that processes one message at a time.
- If the number of listeners is 2, and additional instances is 1, you get two threads that process messages at the same time.
- If the number of listeners is 2, and additional instances is 0, you get two threads that receive data from the EIS, but only one message flow thread will ever run.

The listeners block processing until a message flow instance is available; the listeners do not queue multiple pieces of work. If you leave the number of listeners set to 1 (the default value), the broker ensures that the number of listeners is equal

to the number of additional instances plus one. Additional threads can increase the throughput of a message flow, but consider the potential effect on message order.

Using configurable services for SAP nodes

SAP nodes can get SAP connection details from either the adapter component or a configurable service. By using a configurable service, you can change the connection details for an adapter without the need to redeploy the adapter. For more details about creating, changing, reporting, and deleting the configurable services for SAP, see "Changing connection details for SAP adapters" on page 299.

Look at the following samples to see how to use this node:

- SAP Connectivity

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

Terminals and properties

When you have put an instance of the SAPInput node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. If you double-click an SAPInput node, you open the Adapter Connection wizard. All mandatory properties for which you must enter a value (those properties that do not have a default value defined) are marked with an asterisk.

The SAPInput node terminals are described in the following table.

Terminal	Description
Out	Business object events from the adapter are sent to the Out terminal.
Failure	If an error occurs in the SAPInput node, the message is propagated to the Failure terminal. Information about the error, and business object events can also be propagated to the Failure terminal.
Catch	Business object events are sent to the Catch terminal if they cause an uncaught exception in the message flow. If the Catch terminal is not connected, the retry process is activated to handle the business object.

The following tables describe the node properties. The columns headed M indicate whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the columns headed C indicate whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The SAPInput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, SAPInput	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The SAPInput node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Adapter component	Yes	Yes		The name of the adapter component that contains configuration properties for the adapter. Either enter a name of an adapter file, or click **Browse** to select an adapter file from the list of files that are available in referenced message set projects.	adapterComponent

The SAPInput node Routing properties are described in the following table.

Property	M	C	Default	Description
Set destination list	No	No	Selected	Specifies whether to add the method binding name to the route-to-label destination list. If you select this check box, the method binding name is added so that you can use a RouteToLabel node in the message flow after the SAPInput node.
Label prefix	No	No		The prefix to add to the method name when routing to a label. Add a label prefix to avoid a clash of corresponding label nodes when you include multiple WebSphere Adapters input nodes in the same message flow. By default, there is no label prefix, so the method name and label name are identical.

The SAPInput node Input Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	DataObject	The domain that is used to parse the incoming message. By default, the message that is propagated from the SAPInput node is in the DataObject domain. You cannot specify a different domain.
Message set	Yes	No	Set automatically	The name of the message set in which the incoming message is defined. This field is set automatically from the Adapter component property. If you set this property, then later update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Message type	No	No		The name of the incoming message. The node detects the message type automatically. You cannot set this property.
Message format	No	No		The name of the physical format of the incoming message. You cannot set this property.

The SAPInput node Transactionality properties are described in the following table.

Property	M	C	Default	Description
Transaction mode	No	No	Yes	The transaction mode on this input node determines whether the rest of the nodes in the message flow are executed under sync point.

The Instances properties of the SAPInput node are described in the following table. For a full description of these properties, refer to "Configurable message flow properties" on page 1324.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Additional instances pool	No	Yes	Use Pool Associated with Message Flow	The pool from which additional instances are obtained. • If you select Use Pool Associated with Message Flow, additional instances are obtained from the message flow value. • If you select Use Pool Associated with Node, additional instances are allocated from the additional instances of the node based on the number specified in the Additional instances property.	componentLevel
Additional instances	No	Yes	0	The number of additional instances that the node can start if the Additional instances pool property is set to Use Pool Associated with Node. By default, no additional instances are given to the node.	additionalInstances

The SAPInput node Retry properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Retry mechanism	No	No	Failure	Specifies how retry processing is handled when a failure is rolled back to the SAPInput node. • If you select Failure, retry processing is not performed so you cannot set the remaining properties. • If you select Short and long retry, retry processing is performed first at the interval that is specified by the Short retry interval property, and if that is unsuccessful, it is then performed at the interval that is specified by the Long retry interval property.	
Retry threshold	No	Yes	0	The maximum number of times that retry processing is performed for short retry.	retryThreshold
Short retry interval	No	Yes	0	The interval between short retry attempts.	shortRetryInterval
Long retry interval	No	Yes	0	The interval between long retry attempts.	longRetryInterval

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

SAPRequest node

Use the SAPRequest node to send requests to an SAP application.

This topic contains the following sections:

Purpose

Use the SAPRequest node to send requests to SAP applications. For example, the SAPRequest node might request information from an SAP Enterprise Information System (EIS). A customer business object is sent to SAP, causing SAP to retrieve information about a customer, such as an address and account details. The response information that is retrieved by the SAPRequest node can then be used by the rest of the message flow. The SAPRequest node can send and receive business data.

The SAPRequest node is contained in the **WebSphere Adapters** drawer of the message flow node palette, and is represented in the workbench by the following icon:

Using the SAPRequest node in a message flow

The SAPRequest node needs an adapter component to function correctly. You set the component by using the Adapter component node property and business object definitions, which are stored in the message set that you reference from the node. For this reason, you must provide a message set. By default, the message that is propagated from the SAPRequest node is in the DataObject domain, so that the Message domain property is set to DataObject. You cannot specify a different domain. The node automatically detects the message type.

The SAPRequest node supports local transactions by using the Local Transaction Manager of the broker.

You can deploy several WebSphere Adapters request nodes that use the same adapter component to an execution group.

You can use the mqsisetdbparms command in the following format to configure an account name with a user name and password for the Adapter for SAP Software. The mqsisetdbparms command stores the password in case-sensitive form. However, when the SAP GUI sets a password, it automatically converts the password to uppercase. Therefore, specify an uppercase password to connect to the SAP system.

```
mqsisetdbparms broker name -n adapter name -u user name -p PASSWORD
```

For example:

```
mqsisetdbparms BRK1 -n eis::SAPCustomerOutbound.outadapter -u sapuid -p ********
```

Look at the following sample to see how to use this node:

- SAP Connectivity

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

Using configurable services for SAP nodes

SAP nodes can get SAP connection details from either the adapter component or a configurable service. By using a configurable service, you can change the connection details for an adapter without the need to redeploy the adapter. For more details about creating, changing, reporting, and deleting the configurable services for SAP, see "Changing connection details for SAP adapters" on page 299.

Terminals and properties

When you have put an instance of the SAPRequest node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. If you double-click an SAPRequest node, you open the Adapter Connection wizard. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The SAPRequest node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts the request business object.
Out	The output terminal to which the response business object is sent if it represents successful completion of the request, and if further processing is required within this message flow.
Failure	If an error occurs in the SAPRequest node, the message is propagated to the Failure terminal. Information about the error, and business object events can also be propagated to the Failure terminal.

The following tables describe the node properties. The columns headed M indicate whether the property is *mandatory* (marked with an asterisk on the panel if you must enter a value when no default is defined); the columns headed C indicate whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The SAPRequest node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, SAPRequest	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The SAPRequest node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Adapter component	Yes	No		The name of the adapter component that contains configuration properties for the adapter. Either enter a name of an adapter file, or click **Browse** to select an adapter file from the list of files that are available in referenced message set projects.	
Default method	Yes	Yes		The default method binding to use.	defaultMethod

The SAPRequest node Response Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	DataObject	The domain that is used to parse the response message. By default, the message that is propagated from the SAPRequest node is in the DataObject domain. You cannot specify a different domain.
Message set	Yes	No	Set automatically	The name of the message set in which the incoming message is defined. This field is set automatically from the Adapter component property. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Message type	No	No		The name of the response message. The node detects the message type automatically. You cannot set this property.
Message format	No	No		The name of the physical format of the response message. You cannot set this property.

The SAPRequest node Transactionality properties are described in the following table.

Property	M	C	Default	Description
Transaction mode	No	No	Automatic	Specifies how updates are handled. • If you select Yes, the SAPRequest node takes part in the local transaction that is started by the message flow's input node. • If you select No, the SAPRequest node does not take part in the local transaction that is started by the message flow's input node. • If you select Automatic, the SAPRequest node uses the value that is set on the input node that drives the message flow. For example, if the message flow is driven by an SAPInput node, the SAPRequest assumes the Transaction mode that is set on the SAPInput node. For more information about transactionality, see "SAP BAPI transaction commit" on page 19.

The SAPRequest node Request properties are described in the following table.

Property	M	C	Default	Description
Method Location	Yes	No	$LocalEnvironment/ Adapter/MethodName	The location of the business method (such as createPurchaseOrder or deletePurchaseOrder) that is used to trigger the SAPRequest node to perform an action on the external system.
Data Location	Yes	No	$Body	The location in the incoming message tree from which data is retrieved to form the request that is sent from the SAPRequest node to the EIS.

The SAPRequest node Result properties are described in the following table.

Property	M	C	Default	Description
Output data location	No	No	$OutputRoot	The message tree location to which the SAPRequest node sends output.

Property	M	C	Default	Description
Copy local environment	No	No	Selected	This property controls how the local environment is copied to the output message. If you select this check box, a new copy of the local environment is created in the tree (at each node in the message flow), and it is populated with the contents of the local environment from the preceding node. Therefore, if a node changes the local environment, the previous nodes in the flow do not see those changes because they have their own copies. This behavior might be an issue if you are using a FlowOrder node, or if you use the propagate command on a Compute node. If you clear the check box, each node does not generate its own copy of the local environment, but it uses the local environment that is passed to it by the previous node. Therefore, if a node changes the local environment, those changes are seen by the upstream nodes.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

SCADAInput node

Use the SCADAInput node to receive messages from clients that connect to the broker across the WebSphere MQ Telemetry Transport.

This topic contains the following sections:

Purpose

SCADA device clients use the MQIsdp protocol to send messages that are converted by the SCADAInput node into a format that is recognized by WebSphere Message Broker. The node also establishes the processing environment for these messages.

Message flows that handle messages received from SCADA devices must always start with a SCADAInput node. Set the SCADAInput node properties to control the way in which messages are received; for example, you can indicate that a message is to be processed under transaction control.

When you deploy message flows that contain SCADA nodes to a broker, deploy them to a single execution group, regardless of the number of message flows.

The execution group to which the SCADA flows are deployed must be the default execution group. The default execution group can be identified by inspecting the

`defaultExecutionGroup` field in the BIP2201 message at the execution group startup. A value of `true` denotes the default execution group.

SCADA is primarily a publish/subscribe protocol; therefore, you typically include a Publication node to end the flow. In scenarios where you do not want to use a Publication node, include a SCADAOutput node. If you include a SCADAOutput node, you must also include a SCADAInput node, regardless of the source of the messages, because the SCADAInput node provides the connectivity information that is required by the SCADAOutput node.

If you include an output node in a message flow that starts with a SCADAInput node, it can be any of the supported output nodes, including user-defined output nodes. You can create a message flow that receives messages from SCADA devices, and generates messages for clients that use all of the supported transports to connect to the broker, because you can configure the message flow to request the broker to provide any necessary conversion.

You can request that the broker starts or stops a SCADA listener by publishing messages with a specific topic. This request can apply to all ports or to a single port that is identified in the message.

The SCADAInput node handles messages in the following message domains:
- MRM
- XMLNSC
- XMLNS
- JMSMap
- JMSStream
- MIME
- BLOB
- XML (this domain is deprecated; use XMLNSC)

You also specify that a user-defined parser is to be used, if appropriate.

z/OS You cannot use SCADAInput nodes in message flows that are to be deployed on z/OS systems.

To process the data in an incoming SCADA message, include a node such as the ResetContentDescriptor node, and set its properties to force the bit stream to be reparsed by a subsequent node.

If you create a message flow to use as a subflow, you cannot use a standard input node; you must use an Input node as the first node to create an In terminal for the subflow.

If your message flow does not receive messages across SCADA connections, choose one of the supported input nodes.

The SCADAInput node is contained in the **Additional Protocols** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

For an example of how to use this node, assume that you create a message flow with a SCADAInput node that receives messages from a remote sensor when it detects a change in its operating environment (for example, a drop in outside temperature). You connect the node to an MQOutput node, which makes these messages available on a queue that is serviced by a WebSphere MQ application that analyses and responds to the information that is received.

In another example, you create a message flow with a SCADAInput node that receives messages every minute from a remote system. The messages contain details of the system's switch settings. The data that is received is fed into a ResetContentDescriptor node to cast the data from binary (BLOB) to MRM message format. The information about the system is stored in a database by using the Database node, and is enriched by using a Compute node to create an XML message, which is published by using a Publication node.

XML messages are expensive to send (because satellite transmission has a high cost for each byte); therefore, it is advantageous to use this method because data is enriched by the broker.

Configuring the SCADAInput node

When you have put an instance of the SCADAInput node into a message flow, you can configure the node. For more information, see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view.

All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

1. Optional: On the **Description** tab, enter a Short description, a Long description, or both. You can also rename the node on this tab.
2. On the **Basic** tab, set the following properties:
   - Update the status of the listener by publishing on the control topic `$SYS/SCADA/MQIsdpListener/<port_number>` with the Payload part of the message set to `ON` or `OFF`. Enable listener on startup is initially selected, which means that the listener for MQIsdp clients is initialized when the message flow is deployed.
   - Specify the Port number on which the MQIsdp server listens. This value must be a unique port number, and must not conflict with other listeners (for example, those set up for WebSphere MQ or WebSphere MQ Everyplace). The default number is 1883.
   - Set the Max threads value to indicate the maximum number of threads available to the MQIsdp server to support clients. The default value is 500.

     If you are using DB2 as your broker database, specify a value that is less than or equal to the value that you have set for the DB2 configuration parameters *maxappls* and *maxagents*. For further information, see Enabling ODBC connections to the databases.
   - Select Use thread pooling if you want the node to use a pool of threads to service clients. If you select this option, the number of threads that are available to the MQIsdp server is limited by Max threads, which is most effective when set to a value between 20 and 40. If you do not select this option, a new thread is created for each client that connects. The check box is cleared initially.

Use this option only if you expect a large number of clients (greater than 200) to connect.

3. On the **Input Message Parsing** tab, set values for the properties that describe the message domain, message set, message type, and message format that the node uses to determine how to parse the incoming message.
   - In Message domain, select the name of the parser that you are using from the list. The default value is BLOB. You can choose from the following options:
     - MRM
     - XMLNSC
     - XMLNS
     - JMSMap
     - JMSStream
     - MIME
     - BLOB
     - XML (this domain is deprecated; use XMLNSC)

     You can also specify a user-defined parser, if appropriate.
   - If you are using the MRM parser or XMLNSC parser in validating mode, select the Message set that you want to use. This list is populated with available message sets when you select MRM or XMLNSC as the domain.
   - If you are using the MRM parser, select the correct message from the list in Message type. This list is populated with messages that are defined in the Message set that you have selected.
   - If you are using the MRM parser, select the format of the message from the list in Message format. This list includes all the physical formats that you have defined for this Message set.
4. On the **Parser Options** subtab:
   - Parse timing is, by default, set to On Demand, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see "Parsing on demand" on page 1449.
   - If you are using the XMLNSC parser, set values for the properties that determine how the XMLNSC parser operates. For more information, see "Manipulating messages in the XMLNSC domain" on page 408.
5. On the **Advanced** tab, set the required value for Transaction mode to define the transactional characteristics of how this message is handled:
   - If you select Automatic, the incoming message is received under sync point if it is marked as persistent; otherwise, it is not received under sync point. The transactionality of any derived messages that are sent subsequently by an output node is determined by the incoming persistence property, unless the output node has overridden transactionality explicitly.
   - If you select Yes, the incoming message is received under sync point. Any derived messages that are sent subsequently by an output node in the same instance of the message flow are sent transactionally, unless the output node has overridden transactionality explicitly.
   - If you select No, the incoming message is not received under sync point. Any derived messages that are sent subsequently by an output node in the flow are sent non-transactionally, unless the output node has specified that the message must be put under sync point.
6. On the **Validation** tab, set the validation properties if you want the parser to validate the body of messages from the Message set. If a message is propagated to the Failure terminal of the node, it is not validated.

For more details, see “Validating messages” on page 204 and “Validation properties” on page 1445.

Connecting the terminals:

The SCADAInput node routes each message that it retrieves successfully to the Out terminal. If this action fails, the message is propagated to the Failure terminal; you can connect nodes to this terminal to handle this condition. If you have not connected the Failure terminal, the message loops continually through the node until the problem is resolved.

If the message is caught by this node after an exception has been thrown further on in the message flow, the message is routed to the Catch terminal. If you have not connected the Catch terminal, the message loops continually through the node until the problem is resolved. Ensure that a node is always connected to this terminal if there is the possibility of the message rolling back within a message flow.

Configuring for coordinated transactions:

When you include a SCADAInput node in a message flow, the value that you set for Transaction mode defines whether messages are received under sync point:

- If you set this property to Yes (the default), the message is received under sync point; that is, within a WebSphere MQ unit of work. Any messages that are sent subsequently by an output node in the same instance of the message flow are put under sync point, unless the output node has overridden this explicitly.
- If you set this property to Automatic, the message is received under sync point if the incoming message is marked as persistent; otherwise, it is not received under sync point. Any message that is sent subsequently by an output node is put under sync point, as determined by the incoming persistence property, unless the output node has overridden this explicitly.
- If you set this property to No, the message is not received under sync point. Any messages that are sent subsequently by an output node in the message flow are not put under sync point, unless an individual output node has specified that the message must be put under sync point.

The MQOutput node is the only output node that you can configure to override this option.

Terminals and properties

The SCADAInput node terminals are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is routed if an error occurs.
Out	The output terminal to which the message is routed if it is successfully retrieved from the queue.
Catch	The output terminal to which the message is routed if an exception is thrown downstream and caught by this node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a

value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The SCADAInput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, SCADAInput	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The SCADAInput node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Enable listener on startup	Yes	No	Selected	This property controls when the listener is started. If you select the check box, the listener starts when the message flow is started by the broker. If you clear the check box, the listener starts on the arrival of a message on the specified port.	
Port	Yes	Yes	1883	The port on which the SCADA protocol is listening.	MQIpdpPort
Max threads	Yes	Yes	500	The maximum number of threads to be started to support SCADA devices.	MQIpdpMaxThreads
Use thread pooling	Yes	Yes	Cleared	If you select the check box, thread pooling is used.	

The SCADAInput node Input Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	BLOB	The domain that is used to parse the incoming message.
Message set	No	No		The name or identifier of the message set in which the incoming message is defined. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Message type	No	No		The name of the incoming message.
Message format	No	No		The name of the physical format of the incoming message.

The SCADAInput node Parser Options properties are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	On Demand	This property controls when an input message is parsed. Valid values are On Demand, Immediate, and Complete. For a full description of this property, see "Parsing on demand" on page 1449.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the XMLNSC parser creates syntax elements in the message tree with data types taken from the XML Schema. You can select this property only if you set the Validate property on the **Validation** tab to Content or Content and Value.
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the message data is displayed under XMLNSC in nodes that are connected to the output terminal when the Input Message Parsing properties Message domain is XMLNS.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in an input message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in an input message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in an input message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the input message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if the Validate property is set to None); entries that are specified in Opaque Elements are ignored if validation is enabled.

The SCADAInput node Advanced property is described in the following table.

Property	M	C	Default	Description
Transaction mode	Yes	No	Yes	This property controls whether the incoming message is received under sync point. Valid values are Automatic, Yes, and No.

The SCADAInput node Validation properties are described in the following table.

For a full description of these properties, see "Validation properties" on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are None, Content and Value, and Content.	validateMaster

Property	M	C	Default	Description	mqsiapplybaroverride command property
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

SCADAOutput node

Use the SCADAOutput node to send a message to a client that connects to the broker using the MQIsdp protocol across the WebSphere MQ Telemetry Transport.

This topic contains the following sections:

Purpose

You use the Publication node to send output to a SCADA client. The SCADAOutput node lets you write your own Publication node.

If you include a SCADAOutput node in a message flow, also include a SCADAInput node, regardless of the source of the messages, because the SCADAInput node provides the connectivity information that is required by the SCADAOutput node.

When you deploy message flows that contain SCADA nodes to a broker, deploy them to a single execution group, regardless of the number of message flows.

The execution group to which the SCADA flows are deployed must be the default execution group. The default execution group can be identified by inspecting the `defaultExecutionGroup` field in the BIP2201 message at the execution group startup. A value of `true` denotes the default execution group.

You cannot use the SCADAOutput node to change the transactional characteristics of the message flow. The transactional characteristics that are set by the message flow's input node determine the transactional behavior of the flow.

z/OS You cannot use SCADAOutput nodes in message flows that you deploy to z/OS systems.

If you create a message flow to use as a subflow, you cannot use a standard output node; use an instance of the Output node to create an out terminal for the subflow through which the message can be propagated.

If you do not want your message flow to send messages to a SCADA device, choose another supported output node.

The SCADAOutput node is contained in the **Additional Protocols** drawer of the message flow node palette, and is represented in the workbench by the following icon:

Connecting the terminals

You cannot include this node in a message flow if you have disabled the publish/subscribe engine of the broker or brokers to which you intend to deploy it. If you deploy a BAR file that includes this message flow, the deployment succeeds, but the broker throws a recoverable exception when a message is received by this node.

Connect the In terminal to the node from which messages that are bound for SCADA destinations are routed.

Connect the Out or Failure terminal of this node to another node in this message flow to process the message further, process errors, or send the message to an additional destination.

If you connect the Out or Failure terminal, the LocalEnvironment that is associated with the message is enhanced with the following information for each destination to which the message has been put by this node:

- Queue name
- Queue manager name
- Message reply identifier (this is set to the same value as message ID)
- Message ID (from the MQMD)
- Correlation ID (from the MQMD)

These values are written in WrittenDestination within the LocalEnvironment tree structure.

If you do not connect either terminal, the LocalEnvironment tree is unchanged.

Terminals and properties

When you have put an instance of the SCADAOutput node into a message flow, you can configure it. For more information, see “Configuring a message flow node” on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The SCADAOutput node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.

Terminal	Description
Failure	The output terminal to which the message is routed if a failure is detected when the message is put to the output queue.
Out	The output terminal to which the message is routed if it has been successfully put to the output queue, and if further processing is required within this message flow.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The SCADAOutput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, SCADAOutput	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Validation properties of the SCADAOutput node are described in the following table.

If a message is propagated to the Failure terminal of the node, it is not validated. For more details, see "Validating messages" on page 204 and "Validation properties" on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	Inherit	This property controls whether validation takes place. Valid values are None, Content and Value, Content, and Inherit.	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

SiebelInput node

Use the SiebelInput node to interact with a Siebel application.

This topic contains the following sections:

Purpose

Use the SiebelInput node to interact with Siebel applications. For example, a SiebelInput node monitors a Siebel system for a specified event. When that event occurs, the SiebelInput node generates a message tree that represents the business object with the new event details. The message tree is propagated to the Out terminal so that the rest of the message flow can use the data to update other systems, or audit the changes.

The SiebelInput node is contained in the **WebSphere Adapters** drawer of the message flow node palette, and is represented in the workbench by the following icon:

Using this node in a message flow

To use the SiebelInput node, you must first create the Siebel event table. For instructions, see "Creating the event store manually" on page 293.

To function correctly, the SiebelInput node needs an adapter component, which you set using the Adapter component node property, and business object definitions, which are stored in the message set that you reference from the node. For this reason, you must provide a message set. By default, the message that is propagated from the SiebelInput node is in the DataObject domain, so the Message domain property is set to DataObject. You cannot specify a different domain. The message type is detected automatically by the node.

The SiebelInput node populates the route to label destination list with the name of the method binding. If you add a RouteToLabel node to the message flow after the SiebelInput node, the RouteToLabel node can use the name of the method binding to route the message to the correct part of the message flow for processing.

You can deploy only one input node that uses a particular adapter component to an execution group, but you can deploy many input nodes that use different adapter components to an execution group.

You can use the mqsisetdbparms command in the following format to configure an account name with a user name and password for the Adapter for Siebel Business Applications.

```
mqsisetdbparms broker name -n adapter name -u user name -p password
```

For example:

```
mqsisetdbparms BRK1 -n eis::SiebelCustomerInbound.inadapter -u siebeluid -p ********
```

Using configurable services for Siebel nodes

Siebel nodes can get Siebel connection details from either the adapter component or a configurable service. By using a configurable service, you can change the connection details for an adapter without the need to redeploy the adapter. For more details about creating, changing, reporting, and deleting the configurable services for Siebel, see “Changing connection details for Siebel adapters” on page 300.

Terminals and properties

When you have put an instance of the SiebelInput node into a message flow, you can configure it; see “Configuring a message flow node” on page 276. The properties of the node are displayed in the Properties view. If you double-click a SiebelInput node, you open the Adapter Connection wizard. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The SiebelInput node terminals are described in the following table.

Terminal	Description
Out	Business object events from the adapter are sent to the Out terminal.
Failure	If an error happens in the SiebelInput node, the message is propagated to the Failure terminal. Information about the error, and business object events can also be propagated to the Failure terminal.
Catch	Business object events are sent to the Catch terminal if they cause an uncaught exception in the message flow. If the Catch terminal is not connected, the retry process is activated to handle the business object.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The SiebelInput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, SiebelInput.	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The SiebelInput node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Adapter component	Yes	Yes		The name of the adapter component that contains configuration properties for the adapter. Either enter a name of an adapter file or click **Browse** to select an adapter file from the list of files that are available in referenced message set projects.	adapterComponent

The SiebelInput node Routing properties are described in the following table.

Property	M	C	Default	Description
Set destination list	No	No	Selected	This property specifies whether to add the method binding name to the route to label destination list. If you select this check box, the method binding name is added so that you can use a RouteToLabel node in the message flow after the SiebelInput node.
Label prefix	No	No		The prefix to add to the method name when routing to label. Add a label prefix to avoid a clash of corresponding label nodes when you include multiple WebSphere Adapters input nodes in the same message flow. By default, there is no label prefix, so the method name and label name are identical.

The SiebelInput node Input Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	DataObject	The domain that is used to parse the incoming message. By default, the message that is propagated from the SiebelInput node is in the DataObject domain. You cannot specify a different domain.
Message set	Yes	No	Set automatically	The name of the message set in which the incoming message is defined. This field is set automatically from the Adapter component property. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Message type	No	No		The name of the incoming message. The node detects the message type automatically. You cannot set this property.
Message format	No	No		The name of the physical format of the incoming message. You cannot set this property.

The SiebelInput node Transactionality properties are described in the following table.

Property	M	C	Default	Description
Transaction mode	No	No	Yes	The transaction mode on this input node determines whether the rest of the nodes in the flow operate under sync point.

The Instances properties of the SiebelInput node are described in the following table. For a full description of these properties, see "Configurable message flow properties" on page 1324.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Additional instances pool	No	Yes	Use Pool Associated with Message Flow	The pool from which additional instances are obtained. • If you select Use Pool Associated with Message Flow, additional instances are obtained from the message flow value. • If you select Use Pool Associated with Node, additional instances are allocated from the node's additional instances based on the number specified in the Additional instances property.	componentLevel

Property	M	C	Default	Description	mqsiapplybaroverride command property
Additional instances	No	Yes	0	The number of additional instances that the node can start if the Additional instances pool property is set to Use Pool Associated with Node. By default, no additional instances are given to the node.	additionalInstances

The SiebelInput node Retry properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Retry mechanism	No	No	Failure	This property specifies how retry processing is handled when a failure is rolled back to the SiebelInput node. • If you select Failure, retry processing is not performed so you cannot set the remaining properties. • If you select Short and long retry, retry processing is performed first at the interval that is specified by the Short retry interval property, and if that is unsuccessful, it is then performed at the interval that is specified by the Long retry interval property.	
Retry threshold	No	Yes	0	The maximum number of times that retry processing is performed for short retry.	retryThreshold
Short retry interval	No	Yes	0	The interval between short retry attempts.	shortRetryThreshold
Long retry interval	No	Yes	0	The interval between long retry attempts.	longRetryThreshold

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

SiebelRequest node

Use the SiebelRequest node to interact with a Siebel application.

This topic contains the following sections:

Purpose

Use the SiebelRequest node to interact with Siebel applications. For example, a SiebelRequest node requests information from a Siebel Enterprise Information System (EIS). A customer business object is sent to Siebel, causing Siebel to retrieve information about a customer, such as an address and account details. The

response information that is retrieved by the SiebelRequest node can then be used by the rest of the message flow. The SiebelRequest node can send and receive business data.

The SiebelRequest node is contained in the **WebSphere Adapters** drawer of the message flow node palette, and is represented in the workbench by the following icon:

Using this node in a message flow

To function correctly, the SiebelRequest node needs an adapter component, which you set using the Adapter component node property, and business object definitions, which are stored in the message set that you reference from the node. For this reason, you must provide a message set. By default, the message that is propagated from the SiebelRequest node is in the DataObject domain, so the Message domain property is set to DataObject. You cannot specify a different domain. The message type is detected automatically by the node.

The SiebelRequest node supports local transactions using the broker's Local Transaction Manager, and global transactions using the broker's external syncpoint coordinator.

You can deploy several WebSphere Adapters request nodes that use the same adapter component to an execution group.

You can use the mqsisetdbparms command in the following format to configure an account name with a user name and password for the Adapter for Siebel Business Applications.

```
mqsisetdbparms broker name -n adapter name -u user name -p password
```

For example:

```
mqsisetdbparms BRK1 -n eis::SiebelCustomerOutbound.outadapter -u siebeluid -p ********
```

Using configurable services for Siebel nodes

Siebel nodes can get Siebel connection details from either the adapter component or a configurable service. By using a configurable service, you can change the connection details for an adapter without the need to redeploy the adapter. For more details about creating, changing, reporting, and deleting the configurable services for Siebel, see “Changing connection details for Siebel adapters” on page 300.

Terminals and properties

When you have put an instance of the SiebelRequest node into a message flow, you can configure it; see “Configuring a message flow node” on page 276. The properties of the node are displayed in the Properties view. If you double-click a SiebelRequest node, you open the Adapter Connection wizard. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The SiebelRequest node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts the request business object.
Out	The output terminal to which the response business object is sent if it represents successful completion of the request, and if further processing is required within this message flow.
Failure	If an error happens in the SiebelRequest node, the message is propagated to the Failure terminal. Information about the error, and business object events can also be propagated to the Failure terminal.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk on the panel if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The SiebelRequest node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, for example, SiebelRequest	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The SiebelRequest node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Adapter component	Yes	No		The name of the adapter component that contains configuration properties for the adapter. Either enter a name of an adapter file, or click **Browse** to select an adapter file from the list of files that are available in referenced message set projects.	
Default method	Yes	Yes		The default method binding to use.	defaultMethod

The SiebelRequest node Response Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	DataObject	The domain that is used to parse the response message. By default, the response message that is propagated from the SiebelRequest node is in the DataObject domain. You cannot specify a different domain.
Message set	Yes	No	Set automatically	The name of the message set in which the incoming message is defined. This field is set automatically from the Adapter component property. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.

Property	M	C	Default	Description
Message type	No	No		The name of the response message. The node detects the message type automatically. You cannot set this property.
Message format	No	No		The name of the physical format of the response message. You cannot set this property.

The SiebelRequest node Transactionality properties are described in the following table.

Property	M	C	Default	Description
Transaction mode	No	No	No	This property specifies that updates are performed independently, not as part of a local transaction. You cannot change this property.

The SiebelRequest node Request properties are described in the following table.

Property	M	C	Default	Description
Method Location	Yes	No	$LocalEnvironment/ Adapter/MethodName	The location of the business method (such as createPurchaseOrder or deletePurchaseOrder) that is used to trigger the SiebelRequest node to perform an action on the external system.
Data Location	Yes	No	$Body	The location in the incoming message tree from which data is retrieved to form the request that is sent from the SiebelRequest node to the EIS.

The SiebelRequest node Result properties are described in the following table.

Property	M	C	Default	Description
Output data location	No	No	$OutputRoot	The message tree location to which the SiebelRequest node sends output.
Copy local environment	No	No	Selected	This property controls how the local environment is copied to the output message. If you select the check box, at each node in the message flow, a new copy of the local environment is created in the tree, and it is populated with the contents of the local environment from the preceding node. So if a node changes the local environment, the upstream nodes do not see those changes because they have their own copies. This behavior might be an issue if you are using a FlowOrder node, or if you use the propagate command on a Compute node. If you clear the check box, each node does not generate its own copy of the local environment, but it uses the local environment that is passed to it by the previous node. So if a node changes the local environment, those changes are seen by the upstream nodes.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

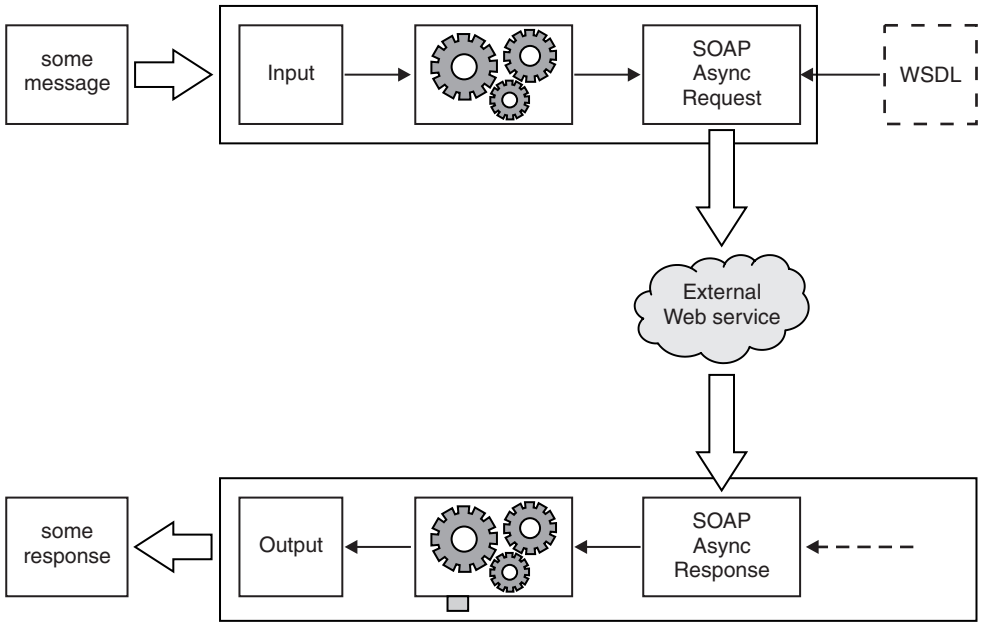
SOAPAsyncRequest node

Use the SOAPAsyncRequest node with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

This topic contains the following sections:

Purpose

The SOAPAsyncRequest node sends a Web service request, but the node does not wait for the associated Web service response to be received. However, the SOAPAsyncRequest node does wait for the HTTP 202 acknowledgment before continuing with the message flow, and the SOAPAsyncRequest node blocks if the acknowledgment is not received. The Web service response is received by the SOAPAsyncResponse node, which can be in a separate message flow. The nodes are used as a pair, and correlate responses against the original requests.



The SOAPAsyncRequest node is the first half of the asynchronous request and response node pair. The SOAPAsyncRequest node calls a remote SOAP-based Web

service. The request is sent by the SOAPAsyncRequest node, but the SOAPAsyncRequest node does not receive the response. The response is received by a SOAPAsyncResponse node that is running on a different thread. The SOAPAsyncResponse node is typically at the beginning of a different message flow; however, it must be in the same execution group as the SOAPAsyncRequest node.

The SOAPAsyncRequest node is WSDL-driven, in a similar manner to the SOAPRequest node.

The SOAPAsyncRequest node is contained in the **Web Services** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

The following sample demonstrates how to use the asynchronous SOAP nodes when you call a Web service. The Web service simulates an order service, and the client shows how existing WebSphere MQ interfaces can be extended to make Web service requests.

- Asynchronous Consumer

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

Configuring the SOAPAsyncRequest node

When you have put an instance of the SOAPAsyncRequest node into a message flow, you can configure it; see “Configuring a message flow node” on page 276. The properties of the node are displayed in the Properties view.

All mandatory properties, for which you must enter a value, are marked with an asterisk.

1. Optional: On the **Description** tab, enter a Short description, a Long description, or both. You can also rename the node on this tab.
2. On the **Basic** tab, you must set the following properties:
   - Unique identifier. You must specify the unique URL fragment that is common to your pair of SOAPAsyncRequest and SOAPAsyncResponse nodes. This property is mandatory.
   - You must associate a WSDL file with this node, and configure a number of WSDL-related properties. Before configuring the WSDL file on this node, you must have a message set with a Deployable WSDL resource.
     – WSDL file name. This property is mandatory and is of type String. If the node was created by dropping a WSDL file from a message set onto the message flow editor, this property is preset to the name of the WSDL file. If the name of the WSDL file is not preset, you can set this property in one of the following ways.
       - If you have Deployable WSDL, you can select from the Deployable WSDL files by clicking **Browse**.
       - If you have WSDL definitions, but no message set, then you can create a message set:

a. Click **Browse** to open the WSDL Selection window.
b. Click **Import/Create New** to open the Import WSDL file wizard.
c. Enter the message set name and message set project name. Click **Next**.
d. Choose the relevant option:
   - If your WSDL file exists in your workspace, select **Use resources from the workspace**, and select the WSDL file.
   - If your WSDL file is in the file system, select **Use external resources**. Select the WSDL file. Click **Next**.
e. Select the WSDL bindings to import. Any warnings or errors are displayed in the wizard banner.
f. Click **Finish**. Result: Creates a new message set project and message set, with message definitions. The WSDL definitions are added to the Deployable WSDL folder.
g. You can now select the WSDL file from the WSDL Selection window. Click **OK**.

- If you have a message set but no WSDL definition, you must generate a WSDL definition; see Generating a WSDL definition from a message set.
- Drag a WSDL file from a message set onto the node.
- Type in a file name that is relative to the message set project in which the deployable WSDL file exists.

When you select a WSDL file for the WSDL file name field, the WSDL is validated to ensure that it is WS-I compliant. The other properties on the **Basic** tab are automatically completed with values based on the WSDL definition.

Only Deployable WSDL can be used to configure the SOAP nodes.

After a valid WSDL file is selected, the message set project to which WSDL file belongs is added as a referenced project to the corresponding flow project, if the reference does not already exist.

If the WSDL file is not valid, or an incorrect file name is entered, an error message is displayed in the Properties view and all WSDL properties are blank.

The following situations lead to error conditions on this property:

- The WSDL file does not come from a message set project, or the WSDL file was not imported correctly; see Importing from WSDL and Importing WSDL definitions from the command line.
- The WSDL file contains no HTTP bindings.
- The WSDL file contains no port type.
- The WSDL file entered in the text box does not exist.

– Port type. This property is mandatory and is of type String. This field lists all the port types defined by the specified WSDL file. By default, the first port type found in the WSDL file that has an associated HTTP binding is selected.

  Error Conditions:

  - Selected Port type does not contain at least one operation.

– Imported binding. This property is mandatory and is of type String. The Imported binding box lists all the SOAP bindings associated with the selected port type. Only HTTP transport is supported. Bindings are listed in the order that they are displayed in the WSDL file. By default, the first

binding that implements the operation and has an associated service port, is selected. This property is updated every time the Port type value changes.

Error Conditions:

- No SOAP bindings (with HTTP transport) in the WSDL file are associated with the Port type.
- Selected binding does not have any operations.

– Binding operation. This property is mandatory and is of type String. The Binding operation box lists all the operations defined by the selected binding. The first operation in the list is selected by default. This property is updated every time the selected binding value changes.
– Service port. This property is mandatory and is of type String. The Service port box lists all the service ports that point to the selected binding. The first service port for the binding is selected by default. This property is updated every time the selected binding value changes.

  Error Conditions:

  - No ports point to the selected binding.

– Target namespace. This property type is String. Target namespace displays the namespace of the selected WSDL file.

When you save the flow file, validation of some of the WSDL-related properties occur:

- It is validated that the WSDL file exists in the message set.
- It is validated that the selected Port type, Binding operation, and Service port are all valid within the content of the selected WSDL file.

If any of these conditions are not met, an error is generated, and you will not be able to add a flow that contains this SOAPAsyncRequest node to the broker archive (BAR) file.

3. On the **HTTP Transport** tab, you can set the HTTP transport related properties:
   - Web service URL. This property is mandatory and is of type String. The Web service URL is automatically derived from the `<soap:address>` element of the selected Service port. Whenever the selected port is updated, the Web service URL is updated accordingly. However, if you override the value, your value persists and the URL is no longer updated from the service port.

     If you choose to override this property you must specify it in the form http://<hostname>[:<port>]/[<path>] where:
     – http://<hostname> must be specified.
     – <port> has a default of 80. If you specify a value, you must include the : before the port number.
     – <path> has a default of forward slash (/). If you specify a value, you must include the forward slash (/) before the path.

     For more details of how to override this property, see Changing the default URL for a SOAPRequest node or a SOAPAsyncRequest node request.
   - Request timeout (in seconds). This property type is Integer. This property has the value of the wait time for the remote server to respond with an acknowledgment that the message has been received. The time in seconds that the node waits for a response from the Web service. The valid range is 1 to (2^{31})-1. You cannot enter a value that represents an unlimited wait. The default is 120.
   - HTTP(S) Proxy Location. This property type is String. In the HTTP(S) Proxy Location field, set the location of the proxy server to which requests are sent. This value must be in the form hostname:port.

- SSL Protocol (if using SSL). This property type is Enumerate. Specify the SSL Protocol that you want to use to make the request. The following options are available:

  SSL This option attempts to connect by using the SSLv3 protocol first, but the handshake can fall back to the SSLv2 protocol where the SSLv2 protocol is supported by the underlying JSSE provider.

  SSLv3 This option attempts to connect with the SSLv3 protocol only. The handshake cannot fall back to SSLv2.

  TLS The default. This option attempts to connect with the TLS protocol only. The handshake cannot fall back to SSLv3 or SSLv2.

  Both ends of an SSL connection must use the same protocol. The protocol must be one that the remote server can accept.
- Allowed SSL Ciphers (if using SSL). This property type is String. This setting enables you to specify a single cipher (such as SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA), or a list of ciphers that are the only ones used by the connection. This list of ciphers must include one or more that are accepted by the remote server. A comma (**,**) is used as a separator between the ciphers. The default value is an empty string, which allows the node to use any, or all, of the available ciphers during the SSL connection handshake. This method enables the greatest scope for making a successful SSL connection.

4. Use the **Advanced** tab to define your headers.

   SOAP headers that are part of the must understand headers list are incorporated into the flow rather than causing a SOAP fault. Adding headers to the must understand headers list stops SOAP faults being generated by SOAP headers.
   - The WSDL-defined SOAP headers table is read-only, and is populated based on the SOAP headers defined in the output part of the selected operations. By default, the check boxes, in the second column of the table, are cleared for all entries in the WSDL-defined SOAP headers table. You must select the relevant check box to add the header to the must understand headers list.
   - You can add custom headers (headers that are not defined in the WSDL file) in the User-defined SOAP headers table. Use **Add**, **Edit**, and **Delete** for this table. You must select the check box, in the second column of the table, to ensure that the newly added custom header is added to the must understand headers list.

   You do not need to add must understand headers for WS-Addressing and WS-Security because these are understood if you configure **WS Extensions**.

   The must understand headers list that is configured on this node is applied to the corresponding SOAPAsyncResponse node when the SOAPAsyncResponse node receives the reply from the remote server.
5. Use the **WS Extensions** tab to configure WS extensions. The tab features two configurations:
   - Use WS-Addressing. This property indicates that WS-Addressing is always engaged on the SOAPAsyncRequest node.
   - WS-Security. The WS-Security table features two columns:
     - Alias
     - XPath Expression

     You can add XPath expressions with an associated Alias value to the WS-Security table. The Alias is resolved in a Policy Set that is created by the

administrator. The Policy Set resolves the Alias to either encrypt or sign the part of the message referred to by the XPath Expression. You can **Add**, **Edit**, and **Delete** in this table.

For more details about WS-Addressing with the SOAPAsyncRequest node, see "WS-Addressing with the SOAPAsyncRequest and SOAPAsyncResponse nodes" on page 754.

6. Use the **Request Parser Options** tab to configure request parser options. The tab features a property that controls whether MTOM is enabled for the parser. Valid values are Yes, No, and Inherit. For more information about MTOM, see "SOAP MTOM" on page 808.

LocalEnvironment overrides

You can dynamically override set values in the LocalEnvironment in the same way as setting values in other elements of a message.

You can set the following property under LocalEnvironment.Destination.SOAP.Request:

Setting	Description
Operation	Overrides the Operation property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Operation = 'myOperation';`
UserContext	You can store context data in the following location in the local environment. The SOAPAsyncResponse node can later retrieve this data. `SET OutputLocalEnvironment.Destination.SOAP.Request.UserContext = 'myData';`

You can set the following properties under LocalEnvironment.Destination.SOAP.Request.Transport.HTTP:

Setting	Description
WebServiceURL	Overrides the Web service URL property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.WebServiceURL = 'http://ibm.com/abc/';`
RequestURI	Overrides the `RequestURI`, which is the path after the URL and port. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.RequestURI = '/abc/def?x=y&g=h';`
Timeout	Overrides the Request timeout (in seconds) property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.Timeout = 42;`
ProxyURL	Overrides the HTTP(S) proxy location property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.ProxyURL = 'my.proxy';`
SSLProtocol	Overrides the SSLProtocol property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.SSLProtocol = 'TLS';` Valid values are: SSL, SSLv3, and TLS.
SSLCiphers	Overrides the Allowed SSL Ciphers (if using SSL) property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.SSLCiphers = 'SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA';`
HTTPVersion	Overrides the `HTTPVersion`. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.HTTPVersion = 'HTTP/1.1';`

Setting	Description
Method	Overrides the Method. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.Method = 'GET';`

ProxyConnectHeaders	Specifies additional headers that are used if the outbound request is an SSL connection through a proxy. These additional headers are sent with the initial CONNECT request to the proxy. For example, you can send proxy authentication information to a proxy server when you are using SSL. You can send multiple headers but each one must be separated by a carriage return and a line feed (ASCII 0x0D 0x0A), in accordance with RFC2616; for example:
	`DECLARE CRLF CHAR CAST(X'0D0A' AS CHAR CCSID 1208);`
	`SET OutputLocalEnvironment.Destination.HTTP.ProxyConnectHeaders =`
	`'Proxy-Authorization: Basic Zm5lcmJsZTpwYXNzd29yZA==' || CRLF ||`
	`'Proxy-Connection: Keep-Alive' || CRLF;`
	
	This setting is used only if the request is an SSL request through a proxy server. To send proxy authentication information for a non-SSL request, specify the individual headers in the HTTPRequestHeader folder, as shown in the following example:
	`SET OutputRoot.HTTPRequestHeader."Proxy-Authorization" =`
	`'Basic Zm5lcmJsZTpwYXNzd29yZA==';`
	`SET OutputRoot.HTTPRequestHeader."Proxy-Connection" = 'Keep-Alive';`

Terminals and properties

The SOAPAsyncRequest node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a SOAP request message for dispatch to the target by the node.
Failure	The output terminal to which the message is routed if a failure is detected when the SOAP request message is dispatched to the target (such as a message validation failure).
Out	The output terminal to which the message is routed if it has been successfully dispatched to the target, and if further processing is required within this message flow. The message that leaves the Out terminal is the same as the message that arrived at the In terminal.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined). The column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The SOAPAsyncRequest node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short description	No	No	None	A brief description of the node.
Long description	No	No	None	Text that describes the purpose of the node in the message flow.

The SOAPAsyncRequest node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Unique identifier	Yes	No		Specify the unique URL fragment that is common to your pair of SOAPAsyncRequest and SOAPAsyncResponse nodes.	asyncResponseCorrelator
WSDL file name	Yes	No		This property type is String. When you select a WSDL file for the WSDL file name field, the WSDL is validated to ensure that it is WS-I compliant. Only Deployable WSDL can be used to configure the SOAP nodes. After a valid WSDL file is selected, the message set project to which WSDL file belongs is added as a referenced project to the corresponding flow project, if the reference does not already exist.	
Port type	Yes	No	By default, the first Port type found in the WSDL file, that has an associated HTTP binding with it, is selected.	This property type is String. This field lists all the port types defined by the specified WSDL file. By default, the first port type found in the WSDL file that has an associated HTTP binding is selected. Error Conditions: • Selected Port type does not contain at least one operation.	
Imported binding	Yes	No		This property type is String. The Imported binding box lists all the SOAP bindings associated with the selected port type. Only HTTP transport is supported. Bindings are listed in the order that they are displayed in the WSDL file. By default, the first binding that implements the operation and has an associated service port, is selected. This property is updated every time the Port type value changes. Error Conditions: • No SOAP bindings (with HTTP transport) in the WSDL file are associated with the Port type. • Selected binding does not have any operations.	
Binding operation	Yes	No		This property type is String. The Binding operation box contains all the operations defined by the selected binding. The first operation in the list is selected by default.	
Service port	Yes	No		This property type is String. The Service port box lists all the service ports that point to the selected binding. The first service port for the binding is selected by default. This property is updated every time the selected binding value changes. Error Conditions: • No ports point to the selected binding.	
Target namespace	No	No		This property type is String. Target namespace displays the namespace of the selected WSDL file.	

The SOAPAsyncRequest node HTTP Transport properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Web service URL	Yes	No		This property type is String. This property is automatically derived from the `<soap:address>` element of the selected Service port. Whenever the selected port is updated, the Web service URL is updated accordingly. However, if you override the value then your value persists and the URL is no longer updated from the service port. If you choose to override this property you must specify it in the form http://<hostname>[:<port>]/[<path>] where: • http://<hostname> must be specified. • <port> has a default of 80. If you specify a value, you must include the colon (:) before the port number. • <path> has a default of forward slash (/). If you specify a value, you must include the forward slash (/) before the path.	webServiceURL
Request timeout (in seconds)	No	Yes	120	This property type is Integer. This property has the value of the wait time for the remote server to respond with an acknowledgment that the message has been received. The time in seconds that the node waits for a response from the Web service. The valid range is 1 to (2^{31})-1. You cannot enter a value that represents an unlimited wait.	requestTimeout
HTTP(S) proxy location	No	Yes		This property type is String. The proxy server to which requests are sent. This value must be in the form hostname:port.	httpProxyLocation
SSL Protocol (if using SSL)	No	Yes	TLS	This property type is Enumerate. The SSL protocol to use when making an HTTPS request. Valid values are: • SSL • SSLv3 • TLS	sslProtocol
Allowed SSL ciphers (if using SSL)	No	Yes	Empty	This property type is String. A comma-separated list of ciphers to use when making an SSL request. The default value of an empty string means use all available ciphers.	allowedSSLCiphers

The SOAPAsyncRequest node Advanced properties are described in the following table.

Property	M	C	Default	Description
WSDL-defined SOAP headers	No	No		The WSDL-defined SOAP headers table is read-only, and is populated based on the SOAP headers defined in the output part of the selected operations. By default, the check boxes, in the second column of the table, are cleared for all entries in the WSDL-defined SOAP headers table. You must select the relevant check box to add the header to the 'must understand headers' list.

Property	M	C	Default	Description
User-defined SOAP headers	No	No		You can add custom headers (headers that are not defined in the WSDL file) in the User-defined SOAP headers table. Use **Add**, **Edit**, and **Delete** for this table. You must select the relevant check box, in the second column of the table, to ensure that the newly added custom header is added to the 'must understand headers' list.

The SOAPAsyncRequest node WS Extensions properties are described in the following table.

Property	M	C	Default	Description
Use WS-Addressing	No	No	Selected	You cannot edit this property. This property indicates that WS-Addressing is always engaged on the SOAPAsyncRequest node.
WS-Security	No	No		This table and features two columns: • Alias • XPath Expression You can add XPath expressions with an associated Alias value to the WS-Security table. The Alias is resolved in a Policy Set that is created by the administrator. The Policy Set resolves the Alias to either encrypt or sign the part of the message referred to by the XPath Expression. You can **Add**, **Edit**, and **Delete** in this table.

The SOAPAsyncRequest node Request Parser Options property is described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Allow MTOM	No	Yes	No	This property controls whether MTOM is enabled for the parser. Valid values are Yes, No, and Inherit. For more information about using SOAP MTOM with the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes; see “Using SOAP MTOM with the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes” on page 748.	allowMTOM

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see “Configuring monitoring event sources using monitoring properties” on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

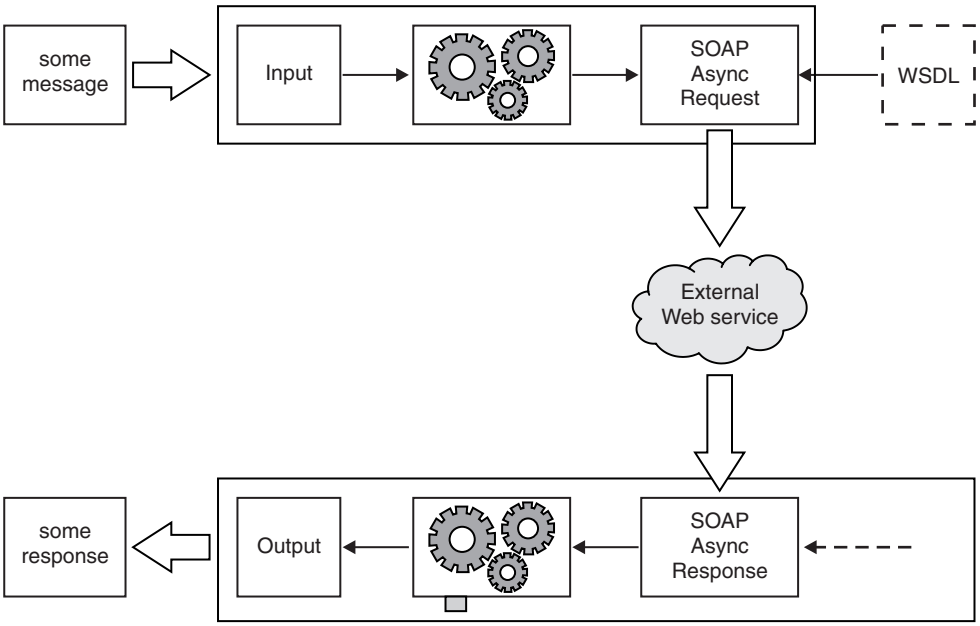
SOAPAsyncResponse node

Use the SOAPAsyncResponse node in conjunction with the SOAPAsyncRequest node to construct a pair of message flows that call a Web service asynchronously.

This topic contains the following sections:

Purpose

The SOAPAsyncRequest node sends a Web service request, but the node does not wait for the associated Web service response to be received. However, the SOAPAsyncRequest node does wait for the HTTP 202 acknowledgment before continuing with the message flow, and the SOAPAsyncRequest node blocks if the acknowledgment is not received. The Web service response is received by the SOAPAsyncResponse node, which can be in a separate message flow. The nodes are used as a pair, and correlate responses against the original requests.



The SOAP parser invokes the XMLNSC parser to parse the XML content of the SOAP Web service, and to validate the XML body of the SOAP Web service. The SOAP parser options are passed through to the XMLNSC parser. For more information, see "Manipulating messages in the XMLNSC domain" on page 408.

The SOAPAsyncResponse node is contained in the **Web Services** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Configuration of the SOAPAsyncResponse node is not WSDL-driven, although the 'must understand headers' list configured on the corresponding SOAPAsyncRequest node is applicable to the SOAPAsyncResponse node.

You can retrieve context data that has been stored by the SOAPAsyncRequest node from the following location in the local environment:

```
LocalEnvironment.SOAP.Response.UserContext
```

The following sample demonstrates how to use the asynchronous SOAP nodes when you call a Web service. The Web service simulates an order service, and the client shows how existing WebSphere MQ interfaces can be extended to make Web service requests.

- Asynchronous Consumer

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

Terminals and properties

When you have put an instance of the SOAPAsyncResponse node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The SOAPAsyncResponse node terminals are described in the following table.

Terminal	Description
Failure	The output terminal to which an asynchronous SOAP response message is routed if a failure is detected when the message received is propagated to the Out flow (such as a message validation failure).
Out	The output terminal to which the asynchronous SOAP response message is routed if it has been successfully received, and if further processing is required in this message flow. If no errors occur in the node, a valid none fault SOAP response message received from an external resource is always sent to the Out terminal first.
Fault	The output terminal to which an asynchronous SOAP fault response is routed if it has been successfully received, and if further processing of the fault is required in this message flow.
Catch	The output terminal to which the message is routed if an exception is thrown downstream and caught by this node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The SOAPAsyncResponse node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type: SOAPAsyncResponse	The name of the node.
Short description	No	No	None	A brief description of the node.
Long description	No	No	None	Text that describes the purpose of the node in the message flow.

The SOAPAsyncResponse node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Unique identifier	Yes	No		Specify the unique URL fragment that is common to your pair of SOAPAsyncRequest and SOAPAsyncResponse nodes.	asyncRequestCorrelator

The SOAPAsyncResponse node Advanced property is described in the following table.

Property	M	C	Default	Description
Set destination list	No	No	Selected	This property indicates whether to add the incoming SOAP operation to the route to label destination list.
Label prefix	No	No		Use this property to add a prefix to the SOAP Operation name in the destination list. You must add a Label prefix if you want to use multiple SOAPAsyncResponse nodes in the same message flow without causing their corresponding Label nodes to clash. By default, the prefix is an empty string so that the operation name and the label name are identical. This property is not available if the Set destination list property is cleared.
Place WS-Addressing headers into LocalEnvironment	No	No	Cleared	This property specifies whether the node puts WS-Addressing headers from the response message into the local environment tree. WS-Addressing headers are not accessible to the flow if this check box is cleared because by default, all headers are processed and removed.

The SOAPAsyncResponse node Instances properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Additional instances pool	No	Yes	Use Pool Associated with Message Flow	The pool from which additional instances are obtained. • If you select Use Pool Associated with Message Flow, additional instances are obtained from the message flow value. • If you select Use Pool Associated with Node, additional instances are allocated from the node's additional instances based on the number specified in the Additional instances property.	componentLevel
Additional instances	No	Yes	0	The number of additional instances that the node can start if the Additional instances pool property is set to Use Pool Associated with Node. By default, no additional instances are given to the node.	additionalInstances

The SOAPAsyncResponse node Response Message Parsing properties are described in the following table. The SOAPAsyncResponse node sets these properties automatically; you cannot set them yourself.

Property	M	C	Default	Description
Message domain	No	No	SOAP	The domain that is used to parse the response message. By default, the message that is propagated from the SOAPAsyncResponse node is in the SOAP domain. You cannot specify a different domain. For more information, see "SOAP parser and domain" on page 96.

Property	M	C	Default	Description
Message set	Yes	No	Set automatically from the WSDL file name property that is provided by the SOAPAsyncRequest node.	The name of the message set in which the response message is defined. Message set is set automatically to the message set that contains the WSDL file that is configured on the corresponding SOAPAsyncRequest node. If you set this property, and then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Message type	No	No		The name of the response message. The node detects the message type automatically. You cannot set this property.
Message format	No	No		The name of the physical format of the response message. You cannot set this property.

The SOAPAsyncResponse node Parser Options properties are described in the following table. The properties are passed through to the XMLNSC parser.

Property	M	C	Default	Description
Parse timing	No	No	On Demand	This property controls when a response message is parsed. Valid values are On Demand, Immediate, and Complete. By default, parse timing is set to On demand, which causes parsing of the input message to be delayed. For a full description of this property, see "Parsing on demand" on page 1449.
Build tree using XML schema data types	No	No	Selected	This property controls whether the XMLNSC parser creates syntax elements in the message tree with data types taken from the XML Schema.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in a response message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in an response message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in a response message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the response message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if Validate is None); entries that are specified in Opaque Elements are ignored if validation is enabled.

The SOAPAsyncResponse node Validation properties are described in the following table. By default, validation is enabled.

If a message is propagated to the Failure terminal of the node, it is not validated. For more details, see "Validating messages" on page 204 and "Validation properties" on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	Content and value	This property controls whether validation takes place. Valid values are None, Content and value, and Content.	validateMaster
Failure action	No	Yes	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and value. Valid values are User trace, Exception list, Local error log, and Exception.	validateFailureAction

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

SOAPEnvelope node

Use the SOAPEnvelope node to add a SOAP envelope onto an existing message. This node is designed to be used with the SOAPExtract node.

This topic contains the following sections:

Purpose

The default behavior of the SOAPEnvelope node is to attach the SOAP envelope from a standard location ($LocalEnvironment/SOAP/Envelope) in the local environment tree; you can specify an explicit location by using an XPath expression.

You can also use the node in a flow without a corresponding SOAPExtract node; the node has an option to create a default SOAP envelope.

The SOAPEnvelope node is contained in the **Web Services** drawer of the palette, and is represented in the workbench by the following icon:

Using the SOAPEnvelope node in a message flow

This node is designed to be used in conjunction with the SOAPExtract node; see "SOAPExtract node" on page 1189.

Configuring the SOAPEnvelope node

When you have put an instance of the SOAPEnvelope node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view.

Supported parsers

This node is designed to work with SOAP messages. Use one of the following parsers:

- XMLNSC
- MRM
- XMLNS

Other XML parsers are not supported because they do not support namespaces. An exception is thrown if a message is received which is not using the correct parser or does not conform to the basic structure of a SOAP message.

Full validation is not done on the SOAP message, which just needs to contain a body element.

As the SOAP domain is not supported by the SOAPEnvelope node, you cannot add the envelope extracted by the SOAPExtract node, from the SOAP domain, back into the message flow again; that is, a flow such as the following example is not supported:

```
SOAPInput node-> SOAPExtract node->SOAPEnvelope node
```

Example SOAP messages

Incoming SOAP envelope

```
<?xml version="1.0"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://ws3.st.mqsi.ibm.com/App/DocLiteral1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Header>
        <tns:requestHeader>
            <tns:assessorUrl>header1</tns:assessorUrl>
        </tns:requestHeader>
    </soapenv:Header>
</soapenv:Envelope>
```

Incoming SOAP message body

```
<?xml version="1.0"?>
<tns:requestAvailability
xmlns:tns="http://ws3.st.mqsi.ibm.com/App/DocLiteral1"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <tns:carDetails>body1</tns:carDetails>
    <tns:claimID>body2</tns:claimID>
    <tns:location>body3</tns:location>
    <tns:reqDate>body4</tns:reqDate>
</tns:requestAvailability>
```

Outgoing SOAP message

```
<?xml version="1.0"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://ws3.st.mqsi.ibm.com/App/DocLiteral1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Header>
        <tns:requestHeader>
            <tns:assessorUrl>header1</tns:assessorUrl>
        </tns:requestHeader>
    </soapenv:Header>
    <soapenv:Body>
        <tns:requestAvailability>
            <tns:carDetails>body1</tns:carDetails>
            <tns:claimID>body2</tns:claimID>
            <tns:location>body3</tns:location>
            <tns:reqDate>body4</tns:reqDate>
        </tns:requestAvailability>
    </soapenv:Body>
</soapenv:Envelope>
```

Terminals and properties

The terminals of the SOAPEnvelope node are described in the following table:

Terminal	Description
In	The input terminal that accepts a SOAP message for processing by the node.
Out	The output terminal that outputs the SOAP message that was constructed from the SOAP message body and a SOAP envelope.
Failure	The output terminal to which the message is routed if a failure is detected during processing.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the SOAPEnvelope node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the SOAPEnvelope node are described in the following table.

Property	M	C	Default	Description
Create new envelope	No	No	Cleared	This property controls whether the node creates a SOAP envelope, or gets an existing one from the message tree. If you select the check box, the node creates a new envelope. If you clear the check box, the node copies the envelope from the value entered in the Existing Envelope Location property.
Existing Envelope Location	No	No	$LocalEnvironment/ SOAP/Envelope	An XPath expression that represents the location from which the node will copy the SOAP envelope. The following correlation names are available: **$Root** The root of the message tree. **$Body** The last child of the root of the message tree (equivalent to /). **$LocalEnvironment** The root of the local environment tree. **$Environment** The root of the global environment tree.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

SOAPExtract node

Use the SOAPExtract node to remove SOAP envelopes, allowing just the body of a SOAP message to be processed. It can also route a SOAP message based on its operation name. Both functions are optional; they are contained in one node because they are often used together.

This topic contains the following sections:

Purpose

The SOAPExtract node can perform two functions:

Extract function
The default behavior is to detach the SOAP envelope to a standard location ($LocalEnvironment/SOAP/Envelope) in the LocalEnvironment tree.

Alternatively, you can specify an explicit location using an XPath expression. Any existing SOAP envelope at the chosen location is replaced.

Routing function
The SOAP message is routed to a Label node in the message flow as identified by the SOAP operation in the message. The SOAP Operation is identified in the SOAP body tag.

Ensure that the message parser options in the properties folder of the outgoing message are correctly set up to parse the message, by copying the message set and message format from the incoming message. The message type is derived from the SOAP envelope message body first child.

Only a single child of the SOAP message body is supported.

The SOAPExtract node is contained in the **Web Services** drawer of the palette, and is represented in the workbench by the following icon:

Using the SOAPExtract node in a message flow

Look at the following sample to see how to use this node:

- SOAP Nodes

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

Configuring the SOAPExtract node

When you have put an instance of the SOAPExtract node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view.

1. Optional: On the **Description** tab, enter a Short description, a Long description, or both. You can also rename the node on this tab.
2. On the **Basic** tab:
   a. Specify in Remove envelope whether the node must remove the soap envelope and place it in the location given in Envelope Destination, or leave it on the message. The default value is that the node removes the envelope.
   b. In Envelope Destination, enter an XPath expression that represents the destination to which the node will copy the envelope. By default, the node copies the envelope to the LocalEnvironment ($LocalEnvironment/SOAP/Envelope).
   c. In Destination path mode, specify the behavior of the Envelope Destination property.
      - Create path: The node creates the tree if the path specifies a location that does not already exist. Only simple expressions of the form `aaa/bbb/ccc` in Envelope Destination are supported. The default.
      - XPath location of existing element: If you know that the destination element exists, you can use any valid XPath 1.0 expression in Envelope Destination.

d. In Route to 'operation' label, specify whether the node must route the message to the SOAP operation given in the message. The default setting is for the node to send the message to the Out terminal.
e. In Label Prefix, enter the value to prefix to the label used for routing by the node. Entering a prefix allows for name spacing between subflows. By default, no value is prefixed to the label name used for routing the message.

Supported parsers

This node is designed to work with SOAP messages. Use one of the following parsers:

- SOAP
- XMLNSC
- MRM
- XMLNS

Other XML parsers are not supported because they do not support namespaces. An exception is thrown if a message is received which is not using the correct parser or does not conform to the basic structure of a SOAP message.

Full validation is not done on the SOAP message, which just needs to contain a body element.

Terminals and properties

The terminals of the SOAPExtract node are described in the following table:

Terminal	Description
In	The input terminal that accepts a SOAP message for processing by the node.
Out	The output terminal that outputs the SOAP message body (without the envelope if the Remove envelope check box is selected on the node properties).
Failure	The output terminal to which the message is routed if a failure is detected during processing.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the SOAPExtract node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the SOAPExtract node are described in the following table.

Property	M	C	Default	Description
Remove envelope	No	No	Selected	If you select the check box, the node removes the SOAP header from the message. For a SOAP tree, the node outputs to the Out terminal the first child of SOAP.body from the SOAP tree. It outputs to Envelope Destination the full SOAP tree minus the first child of SOAP.body. If you clear the check box, the node leaves the envelope on the message. In the case of a SOAP tree, the full tree is propagated to the Out terminal.
Envelope Destination	No	No	$LocalEnvironment/ SOAP/Envelope	An XPath expression that represents the destination to which the node will copy the SOAP envelope. The following correlation names are available: **$Root** The root of the message tree. **$Body** The last child of the root of the message tree (equivalent to /). **$LocalEnvironment** The root of the LocalEnvironment tree. **$Environment** The root of the Global Environment tree.
Destination path mode	No	No	Create path	This determines the behavior of the Envelope Destination property. Set this property: **Create path** The default. The tree is created if the path specifies a location that does not exist. Only simple expressions of the form `aaa/bbb/ccc` are supported. **XPath location of existing element** If you know that the destination element exists, you can enter any valid XPath 1.0 expression.
Route to 'operation' label	No	No	Cleared	This property determines whether the node must route the message to the SOAP operation given in the message. If you select the check box, the message is routed to a Label node that matches the SOAP operation. An exception is thrown if no Label node matches. The name of the first child element of the SOAP body is used to determine the RouteToLabel name. For the 'RPC literal' and 'wrapped doc literal' WSDL types, this is the 'operation' name. For a SOAP tree, the first child of SOAP.Body supplies the operation name. If you clear the check box, the node sends the message to the Out terminal.
Label Prefix	No	No		The value to prefix to the label that the node uses for routing. Entering a prefix allows for name spacing between subflows.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Example SOAP messages

Incoming SOAP message

```
<?xml version="1.0"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://ws3.st.mqsi.ibm.com/App/DocLiteral1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Header>
    <tns:requestHeader>
        <tns:assessorUrl>header1</tns:assessorUrl>
    </tns:requestHeader>
    </soapenv:Header>
    <soapenv:Body>
        <tns:requestAvailability>
            <tns:carDetails>body1</tns:carDetails>
            <tns:claimID>body2</tns:claimID>
            <tns:location>body3</tns:location>
            <tns:reqDate>body4</tns:reqDate>
        </tns:requestAvailability>
    </soapenv:Body>
</soapenv:Envelope>
```

De-enveloped message

The operation name is requestAvailability. Note that the namespacing is removed from the operation.

```
<?xml version="1.0"?>
<tns:requestAvailability
xmlns:tns="http://ws3.st.mqsi.ibm.com/App/DocLiteral1"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <tns:carDetails>body1</tns:carDetails>
    <tns:claimID>body2</tns:claimID>
    <tns:location>body3</tns:location>
    <tns:reqDate>body4</tns:reqDate>
</tns:requestAvailability>
```

Removed envelope

```
<?xml version="1.0"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://ws3.st.mqsi.ibm.com/App/DocLiteral1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Header>
        <tns:requestHeader>
```

```
            <tns:assessorUrl>header1</tns:assessorUrl>
        </tns:requestHeader>
    </soapenv:Header>
</soapenv:Envelope>
```

SOAPInput node

Use the SOAPInput node to process client SOAP messages, so that the broker operates as a SOAP Web Services provider.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow"
- "Using port numbers when deploying a SOAPInput node to an execution group"
- "Connecting the terminals" on page 1195
- "Terminals and properties" on page 1195

Purpose

The SOAPInput node is typically used with the SOAPReply node, which can be included in the same message flow, or a different flow in the same execution group.

You can connect a SOAPReply node to the Out terminal to handle successful responses.

The SOAPInput node is contained in the **Web Services** drawer of the message flow node palette, and is represented in the workbench by the following icon:

Using this node in a message flow

The SOAPInput node can be used in a message flow that accepts and processes SOAP messages. The node is configured using deployable WSDL. Look at the following sample to see how to use this node:

- SOAP Nodes

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

Using port numbers when deploying a SOAPInput node to an execution group

Each execution group that contains a SOAPInput node has one listener and two ports, an HTTP port and an HTTPS port. The default SOAP node port numbers are 7800 for HTTP and 7843 for HTTPS. If you deploy the flow to multiple execution groups, the port number is incremented by one for each successive deployment. The message flow that is deployed to the first execution group receives requests on port 7800 (by default), the next one uses port 7801, and so on, up to the specified limit of 7842. In this scenario, you typically use an intermediary router that listens on one port, then distributes the requests across the range of ports that you are using.

If you do not want the port to be allocated dynamically, you can define a specific port by using the mqsichangeproperties command. You can also change the default range of port numbers by using this command.

Connecting the terminals

The SOAPInput node routes each message that it retrieves successfully to the Out terminal. If message validation fails, the message is routed to the Failure terminal; you can connect nodes to this terminal to handle this condition. If you have not connected the Failure terminal, the message is discarded, the Maximum client wait time expires, and an error is returned to the client.

If the message is caught by this node after an exception has been thrown further on in the message flow, the message is routed to the Catch terminal. If you have not connected the Catch terminal, the message is discarded, the Maximum client wait time expires, and an error is returned to the client.

If the Maximum client wait time expires, by default, the listener sends a SOAP fault message to the client, indicating that its timeout has expired.

Terminals and properties

The SOAPInput node terminals are described in the following table.

Name	Type	Description
Failure	Output data	The output terminal to which a SOAP message is routed if a failure is detected when the message received is propagated to the Out terminal (such as a message validation failure).
Out	Output data	The output terminal to which the SOAP message is routed when it is received successfully and if further processing is required in this message flow. If no errors occur in the input node, a valid SOAP message that is received from an external resource is always sent to the Out terminal first.
Catch	Output data	The output terminal to which the message is routed if an exception is thrown downstream and is caught by this node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The SOAPInput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type; SOAPInput	The name of the node.
Short description	No	No	None	A brief description of the node.
Long description	No	No	None	Text that describes the purpose of the node in the message flow.

The SOAPInput node Basic properties are described in the following table.

Property	M	C	Default	Description
WSDL file name	Yes	No	None	When you select a WSDL file for the WSDL file name property, the WSDL is validated to ensure that it is WS-I compliant. Only deployable WSDL files can be used to configure the SOAP nodes. After a valid WSDL file is selected, the message set project to which the WSDL file belongs is added as a referenced project to the corresponding message flow project, if the reference does not exist. If the WSDL file is not valid, or an incorrect file name is entered, an error message is displayed in the Properties view and all WSDL properties are blank. The following situations result in an error condition: • The WSDL file does not belong to a message set project, or the WSDL file was not imported correctly; see Importing from WSDL and Importing WSDL definitions from the command line. • The WSDL file contains no HTTP bindings. • The WSDL file contains no port type. • The WSDL file that is specified in the field does not exist. If the node was created by dropping a WSDL file from a message set onto the Message Flow editor, this property is preset to the name of the WSDL file. This property takes a string value.
Port type	Yes	No	The first Port type found in the WSDL file (that has an associated HTTP binding with it).	This property lists all the port types that are defined by the specified WSDL file. By default, the first port type found in the WSDL file that has an associated HTTP or JMS binding is selected. This property takes a string value. The following situation causes an error condition: • The selected Port type does not contain at least one operation. When you save the message flow file, validation of some of the WSDL-related properties occur to ensure that: • The WSDL file exists in the message set. • The selected Port type, Binding operation, and Service port are all valid in the content of the selected WSDL file. If one or more of these conditions are not met, an error is generated, and you cannot add a message flow that contains this SOAPInput node to a broker archive (BAR) file.
Imported binding	Yes	No		The Imported binding property lists the imported SOAP bindings associated with the selected port type. Only HTTP transport is supported. When you select a binding, the property tab for the associated transport is enabled; otherwise, it is disabled. Bindings are listed in the order in which they are displayed in the WSDL file. By default, the first imported binding that implements the operation, and has an associated service port, is selected. This property is updated every time the Port type value changes. This property type is String. The following situations cause an error condition: • No imported SOAP bindings (with HTTP transport) in the WSDL file are associated with the Port type. • The selected binding does not have any operations.

Property	M	C	Default	Description
Service port	Yes	No		The Service port field lists all the service ports that point to the selected binding. By default, the first service port for the binding is selected. This property is updated every time the selected binding value changes. This property type is String. The following situation causes an error condition: • No ports point to the selected binding.
Target namespace	Yes	No		This property displays the namespace of the selected WSDL file. This property type is String.

The SOAPInput node HTTP Transport properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Path suffix for URL	Yes	Yes	None	Path suffix for URL is the HTTP path selector on which the node accepts inbound messages. This property is set automatically from the `<soap:address>` element of the selected Service port. Whenever the selected port is updated, URL Selector is updated accordingly. However, if you override this value, your value persists, and the URL is no longer updated from the service port. If you choose to override this property, you must specify the [<path>].	urlSelector
Use HTTPS	No	Yes	Cleared	This property type is Boolean and is configured automatically from the `<soap:address>` element of the selected Service port. If the address contains an HTTPS URL, the check box is selected; otherwise it is cleared. However, if you manually override this property value, it is no longer updated from the corresponding service port.	useHTTPS
Maximum client wait time (sec)	Yes	Yes	180	The time that the client waits for a remote server to respond with a "message received" acknowledgment. The valid range is zero (which means a short wait) through (2^{31})-1. This property specifies the maximum length of time that the TCP/IP listener that received the input message from the Web service client waits for a response from a SOAPReply node. If a response is received within this time, the listener propagates the response to the client. If a response is not received in this time, a SOAP fault message is generated indicating that the timeout has expired.	maxClientWaitTime

The SOAPInput node Advanced properties are described in the following table.

Property	M	C	Default	Description
SOAP 1.1 actor (SOAP 1.2 role)	Yes	No	Ultimate Destination (Ultimate Receiver)	Use this property to configure the SOAP actor (SOAP 1.1 protocol) or SOAP role (SOAP 1.2 protocol) that is performed by the SOAPInput node. The default value is Ultimate Destination (Ultimate Receiver). (Ultimate Destination relates to SOAP 1.1 and Ultimate Receiver relates to SOAP 1.2). You can enter any predefined or user-defined value. Predefined roles are specified in the respective SOAP 1.1 or SOAP 1.2 specifications, and are used to process SOAP headers that are targeted at the specific role. If you select empty, an error occurs when the message flow is validated. This property takes a string value.
Set destination list	No	No	Selected	This property specifies whether to add the method binding name to the route-to-label destination list. If you select this check box, the method binding name is added so that you can use a RouteToLabel node in the message flow after the SOAPInput node. This property type is Boolean.
Label prefix	No	No	None	The prefix to add to the method name when routing to label. Add a Label prefix to avoid a clash of corresponding label nodes when you include multiple input nodes in the same message flow. By default, no label prefix exists; therefore, the method name and label name are identical. The default prefix is an empty string so that the operation name and the label name are identical, but the field displays the user instruction: <enter a prefix if required>. This property is enabled only if the setDestinationList property is enabled.
WSDL defined SOAP headers	No	No		This table is read-only and is populated by the SOAP headers that are defined in the output part of the selected operations. The table is updated automatically when the selected operation is updated. By default, the check boxes in the second column of the table are cleared for all entries in the WSDL-defined table. You must select the check boxes for those headers that you want to add to the must understand headers list. SOAP headers that are part of the must understand headers list are incorporated into the flow rather than causing a SOAP fault. Adding headers to the must understand headers list stops SOAP faults being generated by SOAP headers. You do not have to add must understand headers for WS-Addressing and WS-Security because these elements are understood if you configure WS Extensions. This property is generated in the CMF file.
User defined SOAP headers	No	Yes	None	You can add custom headers in this table. Use the **Add**, **Edit** and **Delete** controls to manipulate rows in this table. You must ensure that the check box for the custom header you have added is selected (in the second column of the table), so that the header is added to the must understand headers list. This property is generated in the CMF file.

The SOAPInput node WS Extensions properties are described in the following table.

Property	M	C	Default	Description
Use WS-Addressing	No	No	Cleared	This property indicates whether to engage WS-Addressing on the SOAPInput node. By default, this check box is selected.
Place WS-Addressing headers into LocalEnvironment	No	No	Cleared	This property specifies whether the node puts WS-Addressing headers received in the message into the local environment tree. WS-Addressing headers are not accessible to the flow if this check box is cleared because, by default, all headers are processed and removed.
WS-Security	No	Yes		This complex property is in the form of a table and consists of two columns: • Alias • XPath Expression You can add XPath expressions with an associated Alias value to the WS-Security table. The Alias is resolved in a policy set that is created by the administrator. The policy set resolves the Alias to either encrypt or sign the part of the message that is referenced by the XPath Expression. You can use the **Add**, **Edit** and **Delete** controls in this table.

The SOAPInput node Input Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	SOAP	The domain that is used to parse the incoming message. By default, the message that is propagated from the SOAPInput node is in the SOAP domain. You cannot specify a different domain.
Message set	Yes	No	Set automatically from the WSDL file name property.	The name of the message set in which the incoming message is defined. This value is set automatically to the message set that contains the WSDLfile when the WSDL is associated with the node. If you set this property, then later update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Message type	No	No		The name of the incoming message. The node detects the message type automatically. You cannot set this property.
Message format	No	No		The name of the physical format of the incoming message. You cannot set this property.

The SOAPInput node Parser Options properties are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	On demand	This property controls when an input message is parsed. Valid values are On demand, Immediate, and Complete. The default value, On demand, causes parsing of the message to be delayed. For a full description of this property, see "Parsing on demand" on page 1449.
Build tree using XML Schema data types	No	No	Selected	This property controls whether the syntax elements in the message tree have data types taken from the XML Schema.

Property	M	C	Default	Description
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in an input message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in an input message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in an input message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the input message that are to be parsed opaquely. Opaque parsing is performed only if validation is not enabled (that is, if Validate is None); entries that are specified in Opaque Elements are ignored if validation is enabled.

The SOAPInput node Error Handling property is described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Send failures during inbound SOAP processing to failure terminal	No	Yes	Cleared	Select this check box to send any fault to the Failure terminal during inbound SOAP processing. If this property is selected, in a situation during inbound SOAP processing that results in a SOAP fault, instead of immediately sending the SOAP fault back to the client, the fault is sent to the Failure terminal instead to allow logging and recovery processing. In this situation, an exception list is sent to the Failure terminal with the inbound message as a BLOB. By default, this check box is cleared.	sendProcessingFaultsToFailure

The SOAPInput node Validation properties are described in the following table. See “Validation properties” on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	Content and value	This property controls whether the SOAP parser validates the body of each input message against XML schema generated from the message set. Valid values are None, Content and value, and Content. The SOAP parser invokes the XMLNSC parser to validate the XML body of the SOAP Web Service. If a message is propagated to the Failure terminal of the node, it is not validated. For more details, see “Validating messages” on page 204 and “Validation properties” on page 1445.	validateMaster

Property	M	C	Default	Description	mqsiapplybaroverride command property
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The SOAPInput node Instances properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Additional instances pool	No	Yes	Use Pool Associated with Message Flow	This property specifies whether additional instance threads are allocated from a thread pool for the whole message flow, or from a thread pool for use by that node only. • If you select Use Pool Associated with Message Flow, additional instances are obtained from the message flow value. • If you select Use Pool Associated with Node, additional instances are allocated based on the number specified in the Additional instances property.	
Additional instances	No	Yes	0	The number of additional instances that the node can start if the Additional instances pool property is set to Use Pool Associated with Node. By default, no additional instances are given to the node.	additionalInstances

The SOAPInput node Retry properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Retry mechanism	No	No	Failure	This property specifies how retry processing is handled when a failure is rolled back to the SOAPInput node. • If you select Failure, retry processing is not performed; therefore, you cannot set the remaining properties. • If you select Short and long retry, retry processing is performed first at the interval that is specified by the Short retry interval property, and if that is unsuccessful, it is then performed at the interval that is specified by the Long retry interval property.	
Retry threshold	No	Yes	0	The maximum number of times that retry processing is performed for short retry.	retryThreshold
Short retry interval	No	Yes	0	The interval between short retry attempts.	shortRetryInterval
Long retry interval	No	Yes	0	The interval between long retry attempts.	longRetryInterval

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

SOAPReply node

Use the SOAPReply node to send SOAP messages from the broker to the originating client in response to a message received by a SOAPInput node.

This topic contains the following sections:

Purpose

The SOAPReply node is typically used with the SOAPInput node, which can be included in the same message flow, or a different flow in the same execution group.

The SOAPReply node is contained in the **Web Services** drawer of the message flow node palette, and is represented in the workbench by the following icon:

Using this node in a message flow

The SOAPReply node can be used in any message flow that needs to send SOAP messages from the broker to the originating client in response to a message received by a SOAPInput node.

Look at the following sample to see how to use this node:

- SOAP Nodes

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

Working with WrittenDestination data

After the reply has been made, the WrittenDestination folder in the LocalEnvironment is updated if WS-Addressing is in use, and with transport details if WS-Addressing is in non-anonymous mode. A WrittenDestination for a SOAPReply node has the following format:

```
WrittenDestination = (
   SOAP  = (
      Reply = (
         WSA = (
            To = 'URI'
```

```
            MessageID = 'id'
            Action = 'doAllTheStuff'
          )      )
   )
)
```

LocalEnvironment overrides

You can dynamically override set values in the local environment in the same way as setting values in other elements of a message. For a full list of values you can override in the local environment, see Local environment overrides.

Terminals and properties

When you have put an instance of the SOAPReply node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The SOAPReply node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if a failure is detected when the message is propagated.
Out	The output terminal to which the message is routed if it has been propagated successfully, and if further processing is required within this message flow.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The SOAPReply node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type: SOAPReply	The name of the node.
Short description	No	No	None	A brief description of the node.
Long description	No	No	None	Text that describes the purpose of the node in the message flow.

The SOAPReply node Validation properties are described in the following table. By default, validation is enabled.

If a message is propagated to the Failure terminal of the node, it is not validated. For more details, see "Validating messages" on page 204 and "Validation properties" on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	Inherit	This property controls whether validation takes place. Valid values are None, Content and Value, Content, and Inherit.	validateMaster
Failure action	No	No	User trace	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are User trace, Local error log, Exception, and Exception list.	

The SOAPReply node Parser Options property is described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Allow MTOM	No	Yes	No	This property controls whether MTOM is enabled for the parser. Valid values are Yes, No, and Inherit. For more information about using SOAP MTOM with the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes, see "Using SOAP MTOM with the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes" on page 748.	caomhinallowMTOM

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

SOAPRequest node

Use the SOAPRequest node to send a SOAP request to the remote Web service. The node is a synchronous request and response node and blocks after sending the request until the response is received.

This topic contains the following sections:

Purpose

The SOAPRequest node is contained in the **Web Services** drawer of the message flow node palette, and is represented in the workbench by the following icon:

Using this node in a message flow

The SOAPRequest node can be used in any message flow that needs to call a Web service. Look at the following sample to see how to use this node:

- SOAP Nodes

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

Terminals and properties

The SOAPRequest node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which a message is routed if a failure is detected when the message is propagated to the Out flow (such as a message validation failure). Failures routed to this terminal include failures caused by the retry processing occurring before the retry propagates the message to the Out flow.
Out	The output terminal to which the message is routed if the SOAP request has been sent and responded to successfully, and if further processing is required within this message flow. If no errors occur within the SOAPRequest node and a none fault SOAP response is received from the external resource it is always sent to the Out terminal first.
Fault	SOAP fault messages received in response to the sent request are directed to the Fault terminal. If no connection is provided to the Fault terminal no further processing occurs for a received fault within this message flow.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined; the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The SOAPRequest node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short description	No	No	None	A brief description of the node.
Long description	No	No	None	Text that describes the purpose of the node in the message flow.

The SOAPRequest node Basic properties are described in the following table.

Property	M	C	Default	Description
WSDL file name	Yes	No	<None>	This property type is String. When you select a WSDL file for the WSDL file name field, the WSDL is validated to ensure that it is WS-I compliant. Only Deployable WSDL can be used to configure the SOAP nodes. After a valid WSDL file is selected, the message set project to which WSDL file belongs is added as a referenced project to the corresponding flow project, if the reference does not exist. If the WSDL file is not valid, or an incorrect file name is entered, an error message is displayed in the Properties view and all WSDL properties are blank. The following situations lead to error conditions on this property: • The WSDL file does not come from a message set project, or the WSDL file was not imported correctly; see Importing from WSDL and Importing WSDL definitions from the command line. • The WSDL file contains no HTTP bindings. • The WSDL file contains no port type. • The WSDL file entered in the text box does not exist. If the node was created by dropping a WSDL file from a message set onto the message flow editor, this property is preset to the name of the WSDL file. If the name of the WSDL file is not preset, you can set this property in one of the following ways. • If you have Deployable WSDL, you can select from the Deployable WSDL files by clicking **Browse**. • If you have WSDL definitions, but no message set, then you can create a message set: 1. Click **Browse** to open the WSDL Selection window. 2. Click **Import/Create New** to open the Import WSDL file wizard. 3. Enter the message set name and message set project name. Click **Next**. 4. Select the relevant option: – If your WSDL file exists in your workspace, select **Use resources from the workspace**, and select the WSDL file. – If your WSDL file is in the file system, select **Use external resources**. Select the WSDL file. Click **Next**. 5. Select the WSDL bindings to import. Any warnings or errors are displayed in the wizard banner. 6. Click **Finish**. Result: Creates a new message set project and message set, with message definitions. The WSDL definitions are added to the Deployable WSDL folder. 7. You can now select the WSDL file from the WSDL Selection window. Click **OK**. • If you have a message set but no WSDL definition, you must generate a WSDL definition. See Generating a WSDL definition from a message set. • Drag a WSDL file from a message set onto the node. • Type in a file name that is relative to the message set project in which the deployable WSDL file exists.

Property	M	C	Default	Description
Port type	Yes	No	By default, the first Port type found in the WSDL file, that has an associated HTTP binding with it, is selected.	This property type is String. The field lists all the Port types defined in WSDL file selected in the WSDL file name property. Error Conditions: • Selected Port type does not contain at least one operation.
Imported binding	Yes	No		This property type is String. This property is updated every time that the Port type value changes. The field lists the imported SOAP bindings with HTTP transport associated with the selected Port type. When you select a binding, the property tab for the associated transport is enabled; otherwise, it is disabled. Bindings are listed in the same order in which they appear in the WSDL file. The selected binding is the one that has both ports and operations. If there is no such binding, then binding with ports is selected. If no bindings have ports then the first binding in the list is selected. Error Conditions: • No SOAP bindings (with HTTP transport) in the WSDL file are associated with the Port type. • The selected binding does not have any operations.
Binding operation	Yes	No		This property type is String. The Binding operation box contains all the operations defined by the selected binding. The first operation in the list is selected by default. This property is updated every time the selected binding value changes
Service port	Yes	No		This property type is String. This field is updated every time that the selected binding is updated. This field lists all the WSDL ports that point to the selected binding. The first service port for the binding is selected by default. This property is updated every time the selected binding value changes. Error Conditions: • No ports point to the selected binding.
Target namespace	Yes	No		Target namespace is implemented as a read-only field. This hidden property type is String. It is updated with the Target namespace of the WSDL file when the WSDL file name is configured.

The SOAPRequest node HTTP Transport properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Web service URL	Yes	Yes	SOAP address of the selected port	The URL of the SOAP address selected. This property is automatically derived from the `<soap:address>` element of the selected Service port. Whenever the selected port is updated, the Web service URL is updated accordingly. However, if you override the value then your value persists and the URL is no longer updated from the service port. If you choose to override this property you must specify it in the form http://<hostname>[:<port>]/[<path>] where: • http://<hostname> must be specified. • <port> has a default of 80. If you specify a value, you must include the colon : before the port number. • <path> has a default of /. If you specify a value, you must include the / before the path. For more details of how to override this property, see Changing the default URL for a SOAPRequest node or a SOAPAsyncRequest node request.	webServiceURL
Request timeout (in seconds)	No	Yes	120	The number of seconds that the client waits for a remote server to respond with a 'message received' acknowledgment. The timeout might take up to one second longer than the value specified.	
HTTP(S) proxy location	No	Yes	Blank	The location of the proxy server to which requests are sent.	httpProxyLocation
Protocol (if using SSL)	No	Yes	TLS	The selected protocol if you use SSL. This property type is Enumerate. The following options are available: **SSL** This option attempts to connect by using the SSLv3 protocol first, but allows the handshake to fall back to the SSLv2 protocol where the SSLv2 protocol is supported by the underlying JSSE provider. **SSLv3** This option attempts to connect with the SSLv3 protocol only. Fallback to SSLv2 is not allowed. **TLS** The default. This option attempts to connect with the TLS protocol only. Fallback to SSLv3 or SSLv2 is not allowed. Both ends of an SSL connection must agree on the protocol to use; therefore, the chosen protocol must be one that the remote server can accept.	sslProtocol

Property	M	C	Default	Description	mqsiapplybaroverride command property
Allowed SSL ciphers (if using SSL)	No	Yes	None	The specific SSL cipher, or ciphers, you are using. This setting allows you to specify a single cipher (such as SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA) or a list of ciphers that are the only ones used by the connection. This set of ciphers must include one or more that are accepted by the remote server. A comma is used as a separator between the ciphers. The default value is an empty string, which allows the node to use any, or all, of the available ciphers during the SSL connection handshake. This method allows the greatest scope for making a successful SSL connection.	allowedSSLCiphers

The SOAPRequest node Advanced properties are described in the following table.

Property	M	C	Default	Description
WSDL-defined SOAP headers	No	No		This table is read-only, and is populated by the SOAP headers defined in the output part of the selected operations. By default, the check boxes, in the second column of the table, are cleared for all entries in the WSDL-defined SOAP headers table. You must select the relevant check box to add the header to the must understand headers list. SOAP headers that are part of the must understand headers list are incorporated into the flow rather than causing a SOAP fault. Adding headers to the must understand headers list stops SOAP faults being generated by SOAP headers. You do not need to add must understand headers for WS-Addressing and WS-Security as they are understood if you configure **WS Extensions**.The table is updated automatically when the selected operation is updated. This property is generated in the CMF file.
User-defined SOAP headers	No	Yes	None	You can add custom headers (headers that are not defined in the WSDL file) in this table. Use **Add**, **Edit**, and **Delete** for this table. You must select the check box, in the second column of the table, to ensure that the newly added custom header is added to the must understand headers list. This property is generated in the CMF file.

The SOAPRequest node WS Extensions properties are described in the following table.

Property	M	C	Default	Description
Use WS-Addressing	No	No		This property specifies whether to use WS-Addressing. For more details about WS-Addressing with the SOAPRequest node, see "WS-Addressing with the SOAPRequest node" on page 754.
Place WS-Addressing headers into LocalEnvironment	No	No	Cleared	This property specifies whether the node puts WS-Addressing headers received in the response message into the local environment tree. WS-Addressing headers are not accessible to the flow if this check box is cleared because by default, all headers are processed and removed.

Property	M	C	Default	Description
WS-Security	No	Yes		This complex property is in the form of a table and consists of two columns: • Alias • XPath Expression You can add XPath expressions with an associated Alias value to the WS-Security table. The Alias is resolved in a Policy Set that is created by the administrator. The Policy Set resolves the Alias to either encrypt or sign the part of the message referred to by the XPath Expression. You can **Add**, **Edit**, and **Delete** in this table.

The SOAPRequest node Response Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	SOAP	The domain that is used to parse the response message. By default, the message that is propagated from the SOAPInput node is in the SOAP domain. You cannot specify a different domain. For more information, see "SOAP parser and domain" on page 96.
Message set	Yes	No	Set automatically from the WSDL file name property.	The name of the message set in which the response message is defined. This property is automatically set to the message set that contains the WSDL file, when the WSDL is associated with the node. If you set this property, then later update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Message type	No	No		The name of the response message. The node detects the message type automatically. You cannot set this property.
Message format	No	No		The name of the physical format of the response message. You cannot set this property.

The SOAPRequest node Parser Options properties are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	On demand	This property controls when a response message is parsed. Valid values are On demand, Immediate, and Complete. The default value, On demand, causes parsing of the message to be delayed. For a full description of this property, see "Parsing on demand" on page 1449.
Build tree using XML schema data types	No	No	Set	This property controls whether the syntax elements in the message tree have data types taken from the XML Schema. The SOAP Parser Options properties determine how the SOAP parser operates. The SOAP parser options are passed through to the XMLNSC parser. For more information, see "Manipulating messages in the XMLNSC domain" on page 408.

Property	M	C	Default	Description
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in a response message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in a response message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in a response message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the input message that are to be opaquely parsed. Opaque parsing is performed only if validation is not enabled (that is, if Validate is None); entries that are specified in Opaque Elements are ignored if validation is enabled.

The SOAPRequest node Validation properties are described in the following table. These properties apply only to the response message; the request message is not validated.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	Content and value	This property controls whether the SOAP parser validates the body of each response message against XML Schema generated from the message. Valid values are None, Content and value, and Content. By default, validation is enabled. The SOAP parser starts the XMLNSC parser to validate the XML body of the SOAP Web service. If a message is propagated to the Failure terminal of the node, it is not validated. For more details, see “Validating messages” on page 204 and “Validation properties” on page 1445.	validateMaster
Failure action	No	Yes	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and value. Valid values are User trace, Local error log, Exception, and Exception list.	validateFailureAction

The SOAPRequest node Request Parser Options property is described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Allow MTOM	No	Yes	No	This property controls whether MTOM is enabled for the parser. Valid values are Yes, No, and Inherit. For more information about using SOAP MTOM with the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes, see "Using SOAP MTOM with the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes" on page 748.	allowMTOM

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Local environment overrides

You can dynamically override set values in the local environment in the same way as setting values in other elements of a message. For a full list of values you can override in the local environment, see Local environment overrides.

Local environment overrides for the SOAPRequest node

You can dynamically override values in the local environment in the same way as setting values in other elements of a message.

Other local environment overrides are available for WS-Addressing. See "WS-Addressing with the SOAPRequest node" on page 754.

You can set the following properties in the SOAPRequest node under `LocalEnvironment.Destination.SOAP.Request`.

Setting	Description
Operation	Overrides the Operation property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Operation = 'myOperation';`

You can set the following HTTP properties in the SOAPRequest node under `LocalEnvironment.Destination.SOAP.Request.Transport.HTTP`.

Setting	Description
WebServiceURL	Overrides the Web service URL property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.WebServiceURL = 'http://ibm.com/abc/';`
RequestURI	Overrides the `RequestURI`, which is the path after the URL and port. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.RequestURI = '/abc/def?x=y&g=h';`
Timeout	Overrides the Request timeout (in seconds) property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.Timeout = 42;`

Setting	Description
ProxyURL	Overrides the HTTP(S) proxy location property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.ProxyURL = 'my.proxy';`
SSLProtocol	Overrides the SSLProtocol property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.SSLProtocol = 'TLS';` Valid values are SSL, SSLv3, and TLS.
SSLCiphers	Overrides the Allowed SSL Ciphers (if using SSL) property on the node. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.SSLCiphers = 'SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA';`
HTTPVersion	Overrides the `HTTPVersion`. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.HTTPVersion = 'HTTP/1.1';`
Method	Overrides the `Method`. For example: `SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.Method = 'GET';`

ProxyConnectHeaders	Specifies additional headers that are used if the outbound request is an SSL connection through a proxy. These additional headers are sent with the initial CONNECT request to the proxy. For example, you can send proxy authentication information to a proxy server when you are using SSL. You can send multiple headers but each one must be separated by a carriage return and a line feed (ASCII 0x0D 0x0A), in accordance with RFC2616; for example:
	`DECLARE CRLF CHAR CAST(X'0D0A' AS CHAR CCSID 1208);`
	`SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.ProxyConnectHeaders = 'Proxy-Authorization: Basic Zm5lcmJsZTpwYXNzd29yZA==' || CRLF ||`
	`'Proxy-Connection: Keep-Alive' || CRLF;`
	
	This setting is used only if the request is an SSL request through a proxy server. To send proxy authentication information for a non-SSL request, specify the individual headers in the HTTPRequestHeader folder, as shown in the following example:
	`SET OutputRoot.HTTPRequestHeader."Proxy-Authorization" = 'Basic Zm5lcmJsZTpwYXNzd29yZA==';`
	`SET OutputRoot.HTTPRequestHeader."Proxy-Connection" = 'Keep-Alive';`

TCPIPClientInput node

Use the TCPIPClientInput node to create a client connection to a raw TCP/IP socket, and to receive data over that connection.

This topic contains the following sections:
- "Purpose"
- "Using the TCPIPClientInput node in a message flow" on page 1216
- "Configuring the TCPIPClientInput node" on page 1216
- "Terminals and properties" on page 1220

Purpose

The TCPIPClientInput node opens connections to a remote server application that is listening on a TCP/IP port. The connections are not made directly by the node but are obtained from a connection pool managed by the WebSphere Message Broker execution group. The execution group uses the default TCPIPClient configurable service to determine which attributes are used for the socket connection. However, if the configurable service is set on the node, the configurable service is used for all the properties, including the host and port number.

When a connection is opened by the connection pool, it is sent to a TCPIPClientInput node (if the Open terminal of the node is connected). The input event is sent to only one TCPIPClientInput node on the connection.

The node requests a client connection that contains data ready for reading. Until such a connection is available, the node is paused, waiting for data (in a similar way to the MQInput node). Therefore, two criteria must be met before the node becomes available:

- A client connection has been made
- At least one byte of data is available to be processed

By default (as set in the configurable service), no client connections are made by the input node. The node relies on the creation of client connections by output or request nodes. In this mode of operation, an input node is never started until an output or request node starts an interaction.

You can change the mode on the configurable service to create a pool of client connections ready for processing. To use this function, minimumConnections must be set to a value larger than zero. The execution group then ensures that the specified number of connections are always available by creating them at the start, and continuing to create the connections until the minimum value is reached.

This behavior is different from the TCPIPServerInput node, which does not attempt to make a minimum number of connections. For more information, see "TCPIPServerInput node" on page 1245.

The client node also has a maximum value, which limits how many connections it can create. More connections than the minimum value can exist as a result of output nodes creating connections.

When connections are available, the second criterion is met when there is at least one byte of data to be processed; otherwise, the connection closes. In either case, the connection is given to the node and the event is processed.

The first record of data is detected in accordance with properties on the node and then sent to the Out terminal. If an error occurs, including a timeout waiting for data or the closure of a connection while waiting for the full record, the data is sent to the Failure terminal. If the connection closes and there is no data, a message is sent to the Close terminal. Although the message has no data, the local environment does have details of the connection that closed.

For both data and close events, the following local environment is created:

Table 14. Location in local environment

Location in local environment	Description
$LocalEnvironment/TCPIP/Input/ConnectionDetails/Type	The client.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/Hostname	The host name used to make a connection.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/Port	The port number used to make a connection.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/OpenTimestamp	The time stamp when the connection was first opened.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/CloseTimestamp	The time stamp when the connection was closed (null if not yet closed).

Table 14. Location in local environment (continued)

Location in local environment	Description
$LocalEnvironment/TCPIP/Input/ConnectionDetails/ SequenceNumber/InputRecord	The sequence number of the message received on this connection. The first record has a sequencing number of 1; the second record has a sequencing number of 2, and so on.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/ SequenceNumber/OutputRecord	The sequence number of the message sent on this connection. The first record has a sequencing number of 1; the second record has a sequencing number of 2, and so on.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/Id	The ID of the socket being used. This ID is an internal identifier used by the message broker to uniquely identify a connection.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/ ReplyId	The reply ID that has been stored on this connection. The value can be any text string.

When the node has constructed the record from the connection stream it releases the connection back to the connection pool for use by other nodes. Properties on the Advanced tab show how that connection can be used by other nodes in the future. By default, the Advanced properties mark the input stream on the TCP/IP connection as being reserved, which means that no other input node can use it, until the message flow's current use is finished. Alternatively, you can reserve the connection until it is unreserved by another node, or not to reserve it at all and permit any other node (or thread in this node) to use the connection straight away. Similar options are available on the output stream but it is kept unreserved by default.

Another node can access a reserved stream only if the ID of the connection is known. This behavior allows all the nodes in a message flow to access the same connection using the same ID while stopping any other flow acquiring the connection.

The TCPIPClientInput node is contained in the **TCPIP** drawer of the palette, and is represented in the workbench by the following icon:

Message structure

The TCPIPClientInput node handles messages in the following message domains:

- MRM
- XMLNSC
- DataObject
- XMLNS
- JMSMap
- JMSStream
- MIME
- BLOB
- XML (this domain is deprecated; use XMLNSC)
- IDOC (this domain is deprecated; use MRM)

Using the TCPIPClientInput node in a message flow

Look at the following samples to see how to use the TCPIPClientInput node:

- TCPIP Client Nodes
- TCPIP Handshake

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

Configuring the TCPIPClientInput node

When you have put an instance of the TCPIPClientInput node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view.

All mandatory properties that do not have a default value defined are marked with an asterisk.

Configure the TCPIPClientInput node:

1. Optional: On the **Description** tab, enter a Short description, a Long description, or both. You can also rename the node on this tab.
2. On the **Basic** tab, set the properties that determine how the TCP/IP connection is controlled.
   - Use the Connection details property to specify either the host name and port number to be used, or the name of a configurable service. This property is mandatory. The following formats are supported:
     - Configurable service name. This value is used to look up the port and host name in configurable services. For example, `TCPIPProfile1`.
     - <Hostname>:<Port>. This value is the host name followed by the port number (separated by a colon). For example, `tcpip.server.com:1111`.
     - <Port>. This value is the port number. In this case, the host name is assumed to be localhost.
   - Use the Timeout waiting for a data record (seconds) property to specify how long the node listens on a connection for more data after the first byte of data has arrived. You can specify any length of time in seconds. The default is 60 seconds. When the specified time has been exceeded, all available data is sent to the Failure terminal.
3. On the **Advanced** tab, set the properties that determine how the data stream is controlled.
   - Use the Close connection property to specify when and how to close the connection.
     - Select No to leave the connection open. This value is the default.
     - Select After timeout to close the connection when a timeout occurs.
     - Select After data has been received to close the connection when the end of the record is found.
     - Select At end of flow to close the connection after the flow has been run.
   - Select Close input stream after a record has been received to close the input stream as soon as the data has been retrieved. When the connection input stream is reserved, no other node can use it without specifying the ID. This property is not selected by default.
   - Use the Input Stream Modification property to specify whether to reserve the input stream for use only by input and receive nodes that specify the

connection ID, or to release the input stream at the end of the flow. These options are available only if you have not selected the Close input stream after a record has been received property.

- Select Leave unchanged to leave the input stream as it was when it entered the node. This value is selected by default.
- Select Reserve input stream (for use by future TCPIP input and receive nodes) to specify that this input stream can be used only by this node and by other receive nodes that request it by specifying the connection ID. When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID.
- Select Reserve input stream (for use by future TCPIP input and receive nodes) then release at end of flow to specify that this input stream can be used only by this node and receive nodes that request it by specifying the correct connection ID. After the flow has been run, this input stream is returned to the pool and becomes available for use by any input or receive node.

- Use the Output Stream Modification property to specify whether to release the output stream.
  - Select Leave unchanged to leave the output stream as it was when it entered the node. This value is selected by default.
  - Select Release output stream and reset ReplyID to specify that this output stream is returned to the pool and is available for use by any output node. The ReplyID is passed in the LocalEnvironment when leaving this node, but is reset for the next record on this connection.

4. On the **Input Message Parsing** tab, set values for the properties that the node uses to determine how to parse the incoming message.

   If the incoming message has an MQRFH2 header, you do not need to set values for the Input Message Parsing properties because the values are derived from the <mcd> folder in the MQRFH2 header; for example:

   ```
   <mcd><Msd>MRM</Msd><Set>DHM4UO906S001</Set><Type>receiptmsg1</Type>
   <Fmt>XML</Fmt></mcd>
   ```

   If you set values, and those values differ from those in the MQRFH2 header, the values in the MQRFH2 header take precedence.

   - In Message domain, select the name of the parser that you are using from the list. The default is BLOB. You can choose from the following options:
     - MRM
     - XMLNSC
     - DataObject
     - XMLNS
     - JMSMap
     - JMSStream
     - MIME
     - BLOB
     - XML (this domain is deprecated; use XMLNSC)
     - IDOC (this domain is deprecated; use MRM)

     You can also specify a user-defined parser, if appropriate.
   - If you are using the MRM or IDOC parser, or the XMLNSC parser in validating mode, select the message set that you want to use. The list contains the message sets that are available when you select MRM, XMLNSC, or IDOC as the domain.

- If you are using the MRM parser, select the correct message type from the list in Message type. This list is populated with available message types when you select the MRM parser.
- If you are using the MRM or IDOC parser, select the correct message format from the list in Message format. This list is populated with available message formats when you select the MRM or IDOC parser.
- Specify the message coded character set ID in Message coded character set ID.
- Select the message encoding from the list in Message encoding or specify a numeric encoding value. The default is Broker System Determined. You can choose from the following options:
  - Little Endian, with IEEE Floating Point (546)
  - Big Endian, with IEEE Floating Point (273)
  - Big Endian, with S390 Floating Point (785)
  - Broker System Determined

  For more information about encoding, see "Converting data with message flows" on page 165.

5. On the **Parser Options** sub-tab:
   - Parse timing is, by default, set to On Demand, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see "Parsing on demand" on page 1449.
   - If you are using the XMLNSC parser, set values for the properties that determine how the XMLNSC parser operates. For more information, see "Manipulating messages in the XMLNSC domain" on page 408.
6. Use the **Retry** tab to define how retry processing is performed when a flow fails. You can set the following retry processing:
   - Retry mechanism determines the action that occurs if the flow fails. The following choices can be set:
     - Select Failure for the node to report a failure without any retry attempts.
     - Select Short retry for the node to retry before reporting a failure if the condition persists. The number of times that it retries is specified in Retry threshold.
     - Select Short retry and long retry for the node to retry, first using the value in Retry threshold as the number of attempts it should make. If the condition persists after the Retry threshold has been reached, the node uses the Long retry interval between attempts.
   - Specify the Retry threshold. The number of times the node retries the flow transaction if the Retry mechanism property is set to either Short retry or Short retry and long retry.
   - Specify the Short retry interval. The length of time, in seconds, to wait between short retry attempts.
   - Specify the Long retry interval. The length of time to wait between long retry attempts until a message is successful, the message flow is stopped, or the message flow is redeployed. The broker property **MinLongRetryInterval** defines the minimum value that the Long retry interval can take. If the value is lower than the minimum, the broker value is used.
7. Use the **Records and Elements** tab to specify how the data is interpreted as records:

- Use the Record detection property to determine how the data is split into records, each of which generates a single message. Choose from the following options:
  - End of stream specifies that all of the data sent in the data stream is a single record.
  - Fixed Length specifies that each record is a fixed number of bytes in length. Each record must contain the number of bytes specified in the Length property, except possibly a shorter final record in the file.
  - Select Delimited if the records you are processing are separated, or terminated, by a DOS or UNIX line end or by a sequence of user-defined delimiter bytes. Specify the delimiter and delimiter type in the Delimiter and Delimiter type properties.
  - Select Parsed Record Sequence if the data contains a sequence of one or more records that are serially recognized by the parser that is specified in Message domain. The node propagates each recognized record as a separate message. If you select this Record detection option, the parser specified in Message domain must be either XMLNSC or MRM (either CWF or TDS physical format).
- If you set Record detection to Fixed Length, use Length to specify the required length of the output record. This value must be between 1 byte and 100 MB. The default is 80 bytes.

  If you set Record detection to Connection closed, Fixed Length, or Delimited, a limit of 100 MB applies to the length of the records. If you set Record detection to Parsed Record Sequence, the TCPIPClientInput node does not determine or limit the length of a record. Nodes that are downstream in the message flow might try to determine the record length, or process a very long record. If you intend to process large records in this way, ensure that your broker has sufficient memory. You might need to apply flow techniques described in the Large Messaging sample to best use the available memory; see Large Messaging.

  You can view samples information only when you use the information center that is integrated with the Message Broker Toolkit or the online information center.
- If you set Record detection to Delimited, use Delimiter to specify the delimiter to be used. Choose from:
  - DOS or UNIX Line End, which, on UNIX systems, specifies the line feed character (<LF>, X'0A'), and, on Windows systems, specifies a carriage return character followed by a line feed character (<CR><LF>, X'0D0A'). The node treats both of these strings as delimiters, irrespective of the system on which the broker is running. If they are both displayed in the same record, the node recognizes both strings as delimiters. The node does not recognize X'15', which, on z/OS systems, is the 'newline' byte; specify a value of Custom Delimiter in this property and a value of 15 in the Custom delimiter property if your input file is coded using EBCDIC new lines, such as EBCDIC files from a z/OS system.
  - Custom Delimiter, which permits a sequence of bytes to be specified in Custom delimiter.
- In Custom delimiter, specify the delimiter byte or bytes to be used when Custom delimiter is set in the Delimiter property. Specify this value as an even-numbered string of hexadecimal digits. The default is X'0A' and the maximum length of the string is 16 bytes (represented by 32 hexadecimal digits).

- If you specify Delimited in Record detection, use Delimiter type to specify the type of delimiter. Permitted values are:
  - Infix. If you select this value, each delimiter separates records. If the data ends with a delimiter, the (zero length) data following the final delimiter is still propagated, although it contains no data.
  - Postfix. If you specify this value, each delimiter terminates records. If the data ends with a delimiter, no empty record is propagated after the delimiter. If the data does not end with a delimiter, it is processed as if a delimiter follows the final bytes of the data. Postfix is the default value.
- The TCPIPClientInput node considers each occurrence of the delimiter in the input as either separating (infix) or terminating (postfix) each record. If the data begins with a delimiter, the node treats the (zero length) contents preceding that delimiter as a record and propagates an empty record to the flow. The delimiter is never included in the propagated message.

8. Use the **Validation** tab to provide validation based on the message set for predefined messages. For more information about validation, see "Validating messages" on page 204. For information about how to complete this tab, see "Validation tab properties" on page 1446.
9. On the **Transactions** tab, set the transaction mode. Although TCP/IP operations are non-transactional, the transaction mode on this input node determines whether the rest of the nodes in the flow are to be executed under point of consistency. Select Yes if you want the flow updates to be treated transactionally (if possible) or No if you do not. The default for this property is No.
10. Optional: On the **Instances** tab, set values for the properties that show the additional instances (threads) that are available for a node. For more details, see "Configurable message flow properties" on page 1324.

Terminals and properties

The terminals of the TCPIPClientInput node are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is routed if an error occurs. This value includes failures caused by retry processing. Even if the Validation property is set, messages propagated to this terminal are not validated.
Out	The output terminal to which the message is routed if it is successfully retrieved from an external resource. If no errors occur within the input node, a message received from an external resource is always sent to the Out terminal first.
Close	The output terminal to which the message is routed if the connection closes.
Catch	The output terminal to which the message is routed if an exception is issued downstream and caught by this node. Exceptions are caught only if this terminal is attached.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file for deployment).

The Description properties of the TCPIPClientInput node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	TCPIPClientInput	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the TCPIPClientInput node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Connection details	Yes	Yes		A string containing either the host name and port number to be used, or the name of a configurable service.	connectionDetails
Timeout waiting for a data record (seconds)	Yes	Yes	60	Specifies how long the node listens on a connection for more data after the first byte of data has arrived. You can specify any length of time in seconds.	timeoutWaitingForData

The Advanced properties of the TCPIPClientInput node are described in the following table.

Property	M	C	Default	Description
Close connection	Yes	No	No	Controls when the connection is closed, or if it remains open. Valid options are: • No • After Timeout • After Data has been Received • At End of Flow
Close input stream after a record has been received	Yes	No	Cleared	Specifies whether to close the input stream as soon as the data has been retrieved. When the connection input stream is reserved, no other node can use it without knowing the ID. This property is not selected by default.
Input Stream Modification	No	No	Leave unchanged	Specifies whether to reserve the input stream for use only by input and receive nodes that specify the connection ID, or to release it at the end of the flow. Valid options are: • Leave unchanged • Reserve input stream (for use by future TCPIP nodes) • Reserve input stream (for use by future TCPIP nodes) then release at end of flow When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID. If the input stream is released at the end of the flow, it is returned to the pool and becomes available for use by any input or receive node.

Property	M	C	Default	Description
Output Stream Modification	No	No	Leave unchanged	Specifies whether this output stream is released and returned to the pool for use by any output node. Valid options are: • Leave unchanged • Release output stream and reset ReplyID If you select Release output stream and reset ReplyID, the ReplyID is passed in the LocalEnvironment when leaving this node, but is reset for the next record on this connection.

The Input Message Parsing properties of the TCPIPClientInput node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Message domain	No	No		The domain that is used to parse the incoming message.	
Message set	No	No		The name or identifier of the message set in which the incoming message is defined. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the Message Set property, or restore the reference to this message set project.	
Message type	No	No		The name of the incoming message.	
Message format	No	No		The name of the physical format of the incoming message.	
Message coded character set ID	Yes	No	Broker System Default	The ID of the coded character set used to interpret the data being read.	messageCodedCharSetIdProperty
Message encoding	Yes	No	Broker System Determined	The encoding scheme for numbers and large characters used to interpret the data being read. Valid values are Broker System Determined or a numeric encoding value. For more information about encoding, see “Converting data with message flows” on page 165.	messageEncodingProperty

The Parser Options properties of the TCPIPClientInput node are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	On Demand	This property controls when an input message is parsed. Valid values are: • On Demand • Immediate • Complete For a full description of this property, see “Parsing on demand” on page 1449.

Property	M	C	Default	Description
Build tree using XML schema data types	No	No	Cleared	This property controls whether the syntax elements in the message tree have data types taken from the XML Schema.
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the message data appears under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or Input Message Parsing property, Message Domain, is XMLNS.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in an input message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in an input message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in an input message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the input message that are to be opaquely parsed by the XMLNSC parser.

The Records and Elements properties of the TCPIPClientInput node are described in the following table:

Property	M	C	Default	Description
Record detection	Yes	No	End of Stream	The mechanism used to identify records in the input data. Valid options are: • End of Stream • Fixed Length • Delimited • Parsed Record Sequence
Length (bytes)	Yes	No	0	The length of each record, in bytes, when Fixed Length record detection is selected.
Delimiter	Yes	No	DOS or UNIX Line End	The type of delimiter bytes that separate, or end, each record when Delimited record detection is selected. Valid options are: • DOS or UNIX Line End • Custom Delimiter (Hexadecimal)
Custom delimiter (hexadecimal)	No	No		The delimiter bytes, expressed in hexadecimal, when Delimited record detection and Custom Delimiter are selected. This property is mandatory only if the Delimiter property is set to Custom Delimiter (Hexadecimal).

Property	M	C	Default	Description
Delimiter type	Yes	No	Postfix	The location of the delimiter when Delimited record detection and Custom Delimiter (Hexadecimal) are selected. Valid options are: • Infix • Postfix This property is ignored unless the Delimiter property is set to Custom Delimiter (Hexadecimal).

The Retry properties of the TCPIPClientInput node are described in the following table:

Property	M	C	Default	Description	mqsiapplybaroverride command property
Retry mechanism	Yes	No	Failure	How the node handles a flow failure. Valid options are: • Failure • Short Retry • Short and Long Retry	
Retry threshold	Yes	Yes	0	The number of times to retry the flow transaction when Retry mechanism is Short retry.	retryThreshold
Short retry interval	No	Yes	0	The interval, in seconds, between each retry if Retry threshold is not zero.	shortRetryInterval
Long retry interval	No	Yes	300	The interval between retries if Retry mechanism is Short and long retry and the retry threshold has been exhausted.	longRetryInterval

The Validation properties of the TCPIPClientInput node are described in the following table.

For a full description of these properties, see "Validation properties" on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are • None • Content and Value • Content	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. Valid values are: • User Trace • Local Error Log • Exception • Exception List	

The Transactions properties of the TCPIPClientInput node are described in the following table:

Property	M	C	Default	Description
Transaction mode	No	Yes	No	The transaction mode on this input node determines whether the rest of the nodes in the flow are executed under point of consistency. Valid options are: • No • Yes

The Instances properties of the TCPIPClientInput node are described in the following table. For a full description of these properties, see "Configurable message flow properties" on page 1324.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Additional instances pool	No	Yes	Use Pool Associated with Message Flow	The pool from which additional instances are obtained. • If you select Use Pool Associated with Message Flow, additional instances are obtained from the message flow pool. • If you select Use Pool Associated with Node, additional instances are allocated from the node's additional instances based on the number specified in the Additional instances property.	componentLevel
Additional instances	No	Yes	0	The number of additional instances that the node can start if the Additional instances pool property is set to Use Pool Associated with Node.	additionalInstances

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

TCPIPClientOutput node

Use the TCPIPClientOutput node to create a client connection to a raw TCP/IP socket, and to send data over that connection to an external application.

This topic contains the following sections:

Purpose

The TCPIPClientOutput node opens connections to a remote server application that is listening on a TCP/IP port. The connections are not made directly by the node

but are obtained from a connection pool managed by the WebSphere Message Broker execution group. The execution group uses the default TCPIPClient configurable service to determine which attributes are used for the socket connection. However, if the configurable service is set on the node, the configurable service is used for all the properties, including the host and port number.

The TCPIPClient configurable service is used to create a pool of client connections ready for processing. To use this function, the minimumConnections property must be set to a value larger than zero. The execution group ensures that the specified number of connections are always available by creating them at the start, and continuing to create the connections until the minimum value is reached.

The node requests a client connection, and, if no connections are available for sending data, the output node requests that the pool creates a new connection. If the maximumConnections property has not been exceeded, a new connection is created.

When the connection has been established, the data is sent. If the data has not been sent successfully within the time limit specified by the node's Timeout sending a data record property, an exception is thrown.

Properties in the local environment can override the TCP/IP connection used by the node:

Table 15. Input local environment properties

Location in local environment	Description
$LocalEnvironment/Destination/TCPIP/Output/Hostname	The host name used to make a connection.
$LocalEnvironment/Destination/TCPIP/Output/Port	The port number used to make a connection.
$LocalEnvironment/Destination/TCPIP/Output/Id	The ID of the socket being used. This value is an internal identifier used by WebSphere Message Broker to uniquely identify a connection.
$LocalEnvironment/Destination/TCPIP/Output/ReplyId	The Reply ID that has been stored on this connection. It can be any text string.

You can dynamically choose the connection details (host name and port number), and the connection used (ID), by using this property. You can also set the Reply ID on the connection. The Reply ID enables a string to be stored in the connection and to be seen in the local environment. You can use this connection to store Reply IDs from other TCPIP nodes or from other transports, such as WebSphere MQ

The output of the node contains the same information as the input, and any fields that were missing from the input are updated with details from the connection used. For example, if the Id property is not provided as input (because you want to create a new connection or reuse a pool connection), the output local environment contains the ID of the connection that is used.

Table 16. Output local environment properties

Location in local environment	Description
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/Hostname	The host name used to make a connection.

Table 16. Output local environment properties (continued)

Location in local environment	Description
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/Port	The port number used to make a connection.
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/OpenTimestamp	The time stamp when the connection was first opened.
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/CloseTimestamp	The time stamp when the connection was closed (null if not yet closed).
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/SequenceNumber	The sequence number of the message received on this connection. The first record has a sequencing number of 1; the second record has a sequencing number of 2, and so on.
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/Id	The ID of the socket being used. This ID is an internal identifier used by WebSphere Message Broker to uniquely identify a connection.
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/ReplyId	The Reply ID that has been stored on this connection. It can be any text string.

If the connection closes (or any other type of exception occurs) while using the TCP/IP transport, an exception is thrown. This exception goes to the Failure terminal if it is connected, otherwise the exception returns back down the flow.

The node also has a Close input terminal. If a message is sent to this terminal, the connection is closed using a combination of the details provided in the node and the local environment.

The TCPIPClientOutput node is contained in the **TCPIP** drawer of the palette and is represented in the workbench by the following icon:

Using the TCPIPClientOutput node in a message flow

The TCPIPClientOutput node can be used in any message flow that needs to send data to an external application. Look at the following samples to see how to use this node:

- TCPIP Client Nodes
- TCPIP Handshake

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

Configuring the TCPIPClientOutput node

When you have put an instance of the TCPIPClientOutput node into a message flow, you can configure it (for more information, see “Configuring a message flow node” on page 276). The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk in that view.

To configure the TCPIPClientOutput node:

1. Optional: On the **Description** tab, enter a short description, a long description, or both. You can also rename the node on this tab.

2. On the **Basic** tab, set the properties that determine how the TCP/IP connection is controlled.
   - Use the Connection details property to specify either the host name and port number to be used, or the name of a configurable service. This property is mandatory. The following formats are supported:
     - Configurable service name. This value is used to look up the port and host name in configurable services. For example, `TCPIPProfile1`.
     - <Hostname>:<Port>. This value is the host name followed by the port number (separated by a colon). For example, `tcpip.server.com:1111`.
     - <Port>. This value is the port number. In this case, the host name is assumed to be localhost.
   - Use the Timeout sending a data record (seconds) property to specify how long the node waits when trying to send data. You can specify any length of time in seconds. When the specified time has been exceeded, all available data is sent to the Failure terminal. The default is 60 seconds.
3. On the **Advanced** tab, set the properties that determine how the data stream is controlled.
   - Use the Send to property to specify whether the data is to be sent to one connection or to all available connections:
     - Select One connection to send the message to only one connection, as specified by the node properties and message overrides. This value is the default.
     - Select All available connections to send the data to all available connections.
   - Use the Close connection property to specify when and how to close the connection.
     - Select No to leave the connection open. This value is the default.
     - Select After timeout to close the connection when a timeout occurs.
     - Select After data has been sent to close the connection when the end of the record has been sent.
   - Select Close output stream after a record has been sent to close the output stream as soon as the data has been sent. This property is not selected by default.
   - Use the Output Stream Modification property to specify whether to reserve or release the output stream. These options are available only if you have not selected the Close output stream after a record has been sent property.
     - Select Leave unchanged to leave the output stream as it was when it entered the node. This value is selected by default.
     - Select Release output stream to specify that this output stream is returned to the pool and is available for use by any output node.
     - Select Reserve output stream (for use by future TCPIP output nodes) to specify that this output stream can be used only by this node and by other output nodes that request it by specifying the connection ID. When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID.
     - Select Reserve output stream (for use by future TCPIP output nodes) then release after propagate to specify that this output stream can be used only by this node and output nodes that request it by specifying the correct connection ID. After the message has been propagated, this output stream is returned to the pool and becomes available for use by any output node.

- Use the Input Stream Modification property to specify whether to reserve the input stream for use only by input and receive nodes that specify the connection ID, or to release it at the end of the flow.
  - Select Leave unchanged to leave the input stream as it was when it entered the node. This value is selected by default.
  - Select Release input stream to specify that this input stream is returned to the pool and is available for use by any input or receive node.
  - Select Reserve input stream (for use by future TCPIP input and receive nodes) to specify that this input stream can be used only by this node and by other input or receive nodes that request it by specifying the connection ID. When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID.
  - Select Reserve input stream (for use by future TCPIP input and receive nodes) then release after propagate to specify that this input stream can be used only by this node and receive nodes that request it by specifying the correct connection ID. After the message has been propagated, this input stream is returned to the pool and becomes available for use by any input or receive node.

4. On the **Request** tab, specify the location of the data to be written. You can specify the properties on this tab as XPath or ESQL expressions. Content Assist is available in the properties pane and also in the XPath Expression Builder, which you can invoke by clicking **Edit** to the right of each property.
   a. In Data location, specify the input data location. This value is the location in the input message tree that contains the record to be written. The default value is $Body, which is the entire message body ($InputRoot.Body).

      When you are specifying this property, and the data in the message tree that it identifies is owned by a model-driven parser (such as the MRM parser or XMLNSC parser), consider the following issues:
      - If you are using MRM CWF format, ensure that the identified message tree exists as a message definition. If this value is defined as a global element only, exceptions BIP5180 and BIP5167 are generated.
      - If you are using MRM TDS format, the serialization of the identified message is successful if the element is defined as a global element or message. However, if the identified field is not found as a global element or message, note that:
        - If this value is a leaf field in the message tree, the field is written as self-defining. No validation occurs even if validation is enabled.
        - If this value is a complex element, an internal exception is generated, BIP5522, indicating that the logical type cannot be converted to a string.
      - If you are using MRM XML, the events are similar as for the MRM TDS format except that, if the field is a complex element, it is written as self-defining.
      - If you use the XMLNSC parser, no validation occurs even if validation is enabled.

   b. In Hostname location, specify the location of the value to override the Hostname set in the Connection details property of the **Basic** tab. If you do not specify a location, the default value is $LocalEnvironment/Destination/TCPIP/Output/Hostname.

c. In Port location, specify the location of the value to override the Port number set in the Connection details property of the **Basic** tab. If you do not specify a location, the default value is $LocalEnvironment/Destination/TCPIP/Output/Port.

d. In ID location, specify the location of the Id of the socket being used. This internal identifier is used by WebSphere Message Broker to uniquely identify a connection. If you do not specify a location, the default value is $LocalEnvironment/Destination/TCPIP/Output/Id.

e. In Reply ID location, specify the location of the Reply ID that is stored on the connection being used. The Reply ID can be used when data is returned in an input node. If you do not specify a location, the default value is $LocalEnvironment/Destination/TCPIP/Output/ReplyId.

5. Use the **Records and Elements** tab to specify how the TCPIPClientOutput node writes the record that is derived from the message.
   - In Record definition, choose from:
     - Record is Unmodified Data to specify that records are left unchanged. This value is the default.
     - Record is Fixed Length Data to specify that records are padded to a specified length if necessary. You specify this length in the Length property. If the record is longer than the value specified in Length, the node generates an exception. Use the Padding byte (hexadecimal) property to specify the byte to be used for padding the message to the required length.
     - Record is Delimited Data to specify that records are separated by a delimiter and accumulated by concatenation. The delimiter is specified by the Delimiter, Custom delimiter, and Delimiter type properties. The file is finished only when a message is received on the Finish File terminal.
   - In Length (bytes), specify the length (in bytes) of records when Record is Fixed Length Data is specified in Record definition. Records longer than this value cause an exception to be issued. This value must be between 1 byte and 100 MB. The default is 80 bytes.
   - When Record is Fixed Length Data is specified in Record definition, use Padding byte (hexadecimal) to specify the byte to be used when padding records to the specified length if they are shorter than this length. Specify this value as 2 hexadecimal digits. The default value is X'20'.
   - In Delimiter, specify the delimiter to be used if you specify Record is Delimited Data in Record definition. Choose from:
     - Broker System Line End to specify that a line end sequence of bytes is used as the appropriate delimiter for the file system on which the broker is to run. This value is the default. For example, on Windows systems, this value is a 'carriage-return, line-feed' pair (X'0D0A'); on UNIX systems, this value is a single 'line-feed' byte (X'0A'); on z/OS systems, this value is a 'newline' byte (X'15').
     - Custom Delimiter (hexadecimal) to specify that the explicit delimiter sequence defined in the Custom delimiter property is to be used to delimit records.
   - In Custom delimiter (hexadecimal), specify the delimiter sequence of bytes to be used to delimit records when Custom Delimiter is specified in the Delimiter property. Specify this value as an even-numbered string of hexadecimal digits. The default is X'0A' and the maximum length of the string is 16 bytes.

- If you specify Record is Delimited Data in Record definition, use Delimiter type to specify how the delimiter is to separate records. Choose from:
  - Postfix to specify that the delimiter is added after each record that is written. This value is the default.
  - Infix to specify that the delimiter is only inserted between any two adjacent records.

6. On the **Validation** tab, specify the parser validation properties of the node. For more information about validation, see "Validating messages" on page 204. For information about how to complete this tab, see "Validation tab properties" on page 1446.

Terminals and properties

The TCPIPClientOutput node terminals are described in the following table.

Terminal	Type	Description
In	Input data	The input terminal that accepts a message for processing by the node.
Close	Input control	The input terminal to which a message is routed when the connection given in the local environment is closed.
Out	Output data	The output terminal to which the message is routed if it is successfully sent to an external resource. The message received on the In terminal is propagated to the Out terminal and is left unchanged except for the addition of status information.
Close	Output control	The output terminal to which a message propagated from the Close input terminal is routed.
Failure	Output data	The output terminal to which the message is routed if a failure is detected in the node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the TCPIPClientOutput node are described in the following table:

Property	M	C	Default	Description
Node name	No	No	TCPIPClientOutput	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the TCPIPClientOutput node are described in the following table:

Property	M	C	Default	Description	mqsiapplybaroverride command property
Connection details	Yes	Yes		A string containing either the host name and port number to be used, or the name of a configurable service.	connectionDetails

Property	M	C	Default	Description	mqsiapplybaroverride command property
Timeout sending a data record (seconds)	Yes	Yes	60	Specifies how long the node waits when attempting to send data. You can specify any length of time in seconds.	timeoutSendingData

The Advanced properties of the TCPIPClientOutput node are described in the following table.

Property	M	C	Default	Description
Close connection	Yes	No	No	Controls when the connection is closed, or if it remains open. Valid options are: • No • After Timeout • After Data has been Sent
Close output stream after a record has been sent	Yes	No	Cleared	Specifies whether to close the output stream as soon as the data has been sent. This property is not selected by default.
Output Stream Modification	No	No	Leave unchanged	Specifies whether to reserve this output stream or release it and return it to the pool for use by any output node. Valid options are: • Leave unchanged • Release output stream • Reserve output stream (for use by future TCPIP nodes) • Reserve output stream (for use by future TCPIP nodes) then release at end of flow
Input Stream Modification	No	No	Leave unchanged	Specifies whether to reserve the input stream for use only by input and receive nodes that specify the connection ID, or to release it at the end of the message flow. Valid options are: • Leave unchanged • Release input stream • Reserve input stream (for use by future TCPIP nodes) • Reserve input stream (for use by future TCPIP nodes) then release at end of flow When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID. If the input stream is released after the message has been propagated, it is returned to the pool and becomes available for use by any input or receive node.
Send to:	Yes	No	One Connection	Specifies whether the data is to be sent to one connection or to available connections. Valid options are: • One Connection • All Available Connections

The Request properties of the TCPIPClientOutput node are described in the following table:

Property	M	C	Default	Description
Data location	Yes	No	$Body	The location in the input message tree containing the record to be written.
Hostname location	Yes	No	$LocalEnvironment/Destination/TCPIP/Output/Hostname	The message element location containing the host name.
Port location	Yes	No	$LocalEnvironment/Destination/TCPIP/Output/Port	The message element location containing the port.
ID location	Yes	No	$LocalEnvironment/Destination/TCPIP/Output/Id	The message element location containing the ID.
Reply ID location	Yes	No	$LocalEnvironment/Destination/TCPIP/Output/ReplyId	The message element location containing the reply ID.

The Records and Elements properties of the TCPIPClientOutput node are described in the following table:

Property	M	C	Default	Description
Record definition	Yes	No	Record is Unmodified Data	This property controls how the records derived from the message are written. Valid options are: • Record is Unmodified Data • Record is Fixed Length Data • Record is Delimited Data
Length (bytes)	Yes	No	0	The required length of the output record. This property applies only when Record is Fixed Length Data is specified in Record definition.
Padding byte (hexadecimal)	Yes	No	20	The two-digit hexadecimal byte to be used to pad short messages when Record is Fixed Length Data is specified in Record definition.
Delimiter	Yes	No	Broker System Line End	The delimiter to be used when Record is Delimited Data is specified in Record definition. Valid options are: • Broker System Line End • Custom Delimiter (Hexadecimal)
Custom delimiter (hexadecimal)	No	No	None	The delimiter byte sequence to be used when Record is Delimited Data is specified in the Record definition property and Custom Delimiter (Hexadecimal) is specified in the Delimiter property.
Delimiter type	Yes	No	Postfix	This property specifies the way in which the delimiters are to be inserted between records when Record is Delimited Data is specified in Record definition. Valid options are: • Infix • Postfix

The Validation properties of the TCPIPClientOutput node are described in the following table.

For a full description of these properties, see “Validation properties” on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	Inherit	This property controls whether validation takes place. Valid values are: • None • Content and Value • Content • Inherit	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are: • User Trace • Local Error Log • Exception • Exception List	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

TCPIPClientReceive node

Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.

This topic contains the following sections:

Purpose

The TCPIPClientReceive node waits for data to be received on a TCP/IP connection, and retrieves the data. If the connection is closed, an exception is thrown.

When a connection is established, the data is sent to the TCPIPClientReceive node. If the TCPIPClientReceive node fails to receive all of the data within the time specified in the Timeout waiting for a data record property, the message is sent to the Timeout terminal; if no Timeout terminal is connected, an exception is thrown.

Properties in the local environment can override the TCP/IP connection used by the node.

Table 17. Input local environment properties

Location in local environment for input to node	Description
$LocalEnvironment//TCPIP/Receive/Hostname	The host name used to make a connection.

Table 17. Input local environment properties (continued)

Location in local environment for input to node	Description
$LocalEnvironment//TCPIP/Receive/Port	The port number used to make a connection.
$LocalEnvironment/TCPIP/Receive/Id	The ID of the socket being used. This ID is an internal identifier used by WebSphere Message Broker to uniquely identify a connection.
$LocalEnvironment/TCPIP/Receive/ReplyId	The Reply ID to be stored on this connection. This ID can then be used when data is returned on an input node. The Reply ID can be any text string.

These properties enable the connection details (host name and port number) and the connection used (ID) to be chosen dynamically. You can also set the Reply ID on the connection, which enables a string to be stored in the connection and to be seen in the local environment of any data that is received back from this connection. You can use this connection to store Reply IDs from other TCPIP nodes or from other transports, such as WebSphere MQ.

When a record has been retrieved, the ConnectionDetails field in the local environment is populated with the details of the connection that is being used.

Table 18. Output local environment properties

Location in local environment for output from node	Description
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/Type	The client.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/Hostname	The host name used to make a connection.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/Port	The port number used to make a connection.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/OpenTimestamp	The time stamp when the connection was first opened.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/CloseTimestamp	The time stamp when the connection was closed (null if not yet closed).
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/SequenceNumber/InputRecord	The sequence number of the message that is received on this connection. The first record has a sequencing number of 1; the second record is 2; and so on.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/SequenceNumber/OutputRecord	The sequence number of the message sent on this connection. The first record has a sequencing number of 1; the second record is 2; and so on.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/Id	The ID of the socket being used. This ID is an internal identifier used by WebSphere Message Broker to uniquely identify a connection.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/ReplyId	The Reply ID that is stored on this connection. This ID can be any text string.

The TCPIPClientReceive node is contained in the **TCPIP** drawer of the palette, and is represented in the workbench by the following icon:

Message structure

The TCPIPClientReceive node handles messages in the following message domains:

- MRM
- XMLNSC
- DataObject
- XMLNS
- JMSMap
- JMSStream
- MIME
- BLOB
- XML (this domain is deprecated; use XMLNSC)
- IDOC (this domain is deprecated; use MRM)

Using the TCPIPClientReceive node in a message flow

Look at the following samples to see how to use the TCPIPClientReceive node:

- TCPIP Client Nodes
- TCPIP Handshake

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

Configuring the TCPIPClientReceive node

When you have put an instance of the TCPIPClientReceive node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view.

All mandatory properties that do not have a default value defined are marked with an asterisk.

Configure the TCPIPClientReceive node:

1. Optional: On the **Description** tab, enter a Short description, a Long description, or both. You can also rename the node on this tab.
2. On the **Basic** tab, set the properties that determine how the TCP/IP connection is controlled.
   - Use the Connection details property to specify either the host name and port number to be used, or the name of a configurable service. This property is mandatory. The following formats are supported:
     - Configurable service name. This value is used to look up the port and host name in configurable services. For example, `TCPIPProfile1`.
     - <Hostname>:<Port>. This value is the host name followed by the port number (separated by a colon); for example, `tcpip.server.com:1111`
     - <Port>. This value is the port number. In this case, the host name is assumed to be localhost.
   - Use the Timeout waiting for a data record (seconds) property to specify how long the node listens on a connection for more data after the first byte of data has arrived. You can specify any length of time in seconds. The default is 60 seconds. When the specified time has been exceeded, all available data is sent to the Failure terminal.

3. On the **Advanced** tab, set the properties that determine how the data stream is controlled.
   - Use the Close connection property to specify when and how to close the connection.
     - Select No to leave the connection open. This value is the default.
     - Select After timeout to close the connection when a timeout occurs.
     - Select After data has been received to close the connection when the end of the record is found.
   - Select Close input stream after a record has been received to close the input stream as soon as the data has been retrieved. By default this property is not selected. When the connection input stream is reserved, no other node can use it without knowing the ID.
   - Use the Input Stream Modification property to specify whether to reserve the input stream for use only by input and receive nodes that specify the connection ID, or to release the input stream at the end of the flow.
     - Select Leave unchanged to leave the input stream as it was when it entered the node. This value is selected by default.
     - Select Release input stream to specify that this input stream is returned to the pool and is available for use by any input or receive node.
     - Select Reserve input stream (for use by future TCPIP input and receive nodes) to specify that this input stream can be used only by this node and by other input or receive nodes that request it by specifying the connection ID. When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID.
     - Select Reserve input stream (for use by future TCPIP input and receive nodes) then release after propagate to specify that this input stream can be used only by this node and receive nodes that request it by specifying the correct connection ID. After the message has been propagated, this input stream is returned to the pool and becomes available for use by any input or receive node.
   - Use the Output Stream Modification property to specify whether to reserve or release the output stream. These options are available only if you have not selected the Close output stream after a record has been sent property.
     - Select Leave unchanged to leave the output stream as it was when it entered the node. This value is selected by default.
     - Select Release output stream to specify that this output stream is returned to the pool and is available for use by any output node.
     - Select Reserve output stream (for use by future TCPIP output nodes) to specify that this output stream can be used only by this node and by other output nodes that request it by specifying the connection ID. When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID.
     - Select Reserve output stream (for use by future TCPIP output nodes) then release after propagate to specify that this output stream can be used only by this node and output nodes that request it by specifying the correct connection ID. After the message has been propagated, this output stream is returned to the pool and becomes available for use by any output node.
4. On the **Request** tab, specify the location of the data to be written. You can specify the properties on this tab as XPath or ESQL expressions. Content Assist is available in the Properties view and also in the XPath Expression Builder, which you can run by clicking **Edit** to the right of each property.

- In Hostname location, specify the location of the value to override the Hostname that is set in the Connection details property of the **Basic** tab. If you do not specify a location, the default value is $LocalEnvironment/TCPIP/Receive/Hostname.
- In Port location, specify the location of the value to override the Port that is set in the Connection details property of the **Basic** tab. If you do not specify a location, the default value is $LocalEnvironment/TCPIP/Receive/Port.
- In ID location, specify the location of the Id of the socket being used. This internal identifier is used by WebSphere Message Broker to uniquely identify a connection. If you do not specify a location, the default value is $LocalEnvironment/TCPIP/Receive/Id.
- In Reply ID location, specify the location of the Reply ID that is stored on the connection that is being used. The Reply ID can be used when data is returned in an input node. If you do not specify a location, the default value is $LocalEnvironment/TCPIP/Receive/ReplyId.

5. On the **Result** tab, set values for the properties that determine where the reply is stored.
   - Use the Output data location property to specify the start location in the output message tree where the parsed elements from the bit string of the message are stored. The default value is `$OutputRoot`.
   - Use the Copy local environment property to specify whether the local environment is copied to the output message.
     - If Copy local environment is selected, a new copy of the local environment is created in the tree, and it is populated with the contents of the local environment from the preceding node. Therefore, if a node changes the local environment, the upstream nodes are not affected by those changes because they have their own copies. This value is the default.
     - If Copy local environment is not selected, the node does not generate its own copy of the local environment, but uses the local environment that is passed to it by the preceding node. Therefore, if a node changes the local environment, the changes are reflected by the upstream nodes.
6. On the **Input Message Parsing** tab, set values for the properties that the node uses to determine how to parse the incoming message.

   If the incoming message has an MQRFH2 header, you do not need to set values for the Input Message Parsing properties because the values are derived from the `<mcd>` folder in the MQRFH2 header; for example:

   ```
   <mcd><Msd>MRM</Msd><Set>DHM4UO906S001</Set><Type>receiptmsg1</Type>
   <Fmt>XML</Fmt></mcd>
   ```

   If you set values, and if they differ from the values in the MQRFH2 header, the values in the MQRFH2 header take precedence.
   - In Message domain, select the name of the parser that you are using from the list. The default is BLOB. You can choose from the following options:
     - MRM
     - XMLNSC
     - DataObject
     - XMLNS
     - JMSMap
     - JMSStream
     - MIME
     - BLOB
     - XML (this domain is deprecated; use XMLNSC)
     - IDOC (this domain is deprecated; use MRM)

You can also specify a user-defined parser, if appropriate.

- If you are using the MRM or IDOC parser, or the XMLNSC parser in validating mode, select the Message set that you want to use. The list contains the message sets that are available when you select MRM, XMLNSC, or IDOC as the domain.
- If you are using the MRM parser, select the correct message type from the list in Message type. This list is populated with available message types when you select the MRM parser.
- If you are using the MRM or IDOC parser, select the correct message format from the list in Message format. This list is populated with available message formats when you select the MRM or IDOC parser.
- Specify the message coded character set ID in Message coded character set ID.
- Select the message encoding from the list in Message encoding or specify a numeric encoding value. For more information about encoding, see "Converting data with message flows" on page 165.

7. On the **Parser Options** sub-tab:
   - Parse timing is, by default, set to On Demand, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see "Parsing on demand" on page 1449.
   - If you are using the XMLNSC parser, set values for the properties that determine how the XMLNSC parser operates. For more information, see "Manipulating messages in the XMLNSC domain" on page 408.
8. Use the **Records and Elements** tab to specify how the data is interpreted as records. Only one record is retrieved each time the TCPIPClientReceive node is invoked; therefore, if the TCP/IP stream contains multiple logical messages, you must invoke the node multiple times to receive all the messages.
   - Use the Record detection property to determine how the data is split into records, each of which generates a single message. Choose from the following options:
     - Connection closed specifies that all of the data sent during a connection is a single record.
     - Fixed Length specifies that each record is a fixed number of bytes in length. Each record contains the number of bytes specified in the Length property, except possibly a shorter final record in the file.
     - Select Delimited if the records that you are processing are separated, or terminated, by a DOS or UNIX line end or by a sequence of user-defined delimiter bytes. Specify the delimiter and delimiter type in the Delimiter and Delimiter type properties.
     - Select Parsed Record Sequence if the data contains a sequence of one or more records that are serially recognized by the parser specified in Message domain. The node propagates each recognized record as a separate message. If you select this Record detection option, the parser specified in Message domain must be either XMLNSC or MRM (either CWF or TDS physical format).
   - If you set Record detection to Fixed Length, use Length to specify the required length of the output record. This value must be between 1 byte and 100 MB. The default is 80 bytes.

     If you set Record detection to Connection closed, Fixed Length, or Delimited, a limit of 100 MB applies to the length of the records. If you set Record detection to Parsed Record Sequence, the TCPIPClientReceive node does not determine or limit the length of a record. Nodes that are downstream in the

message flow might try to determine the record length or process a long record. If you intend to process large records in this way, ensure that your broker has sufficient memory. You might need to apply flow techniques described in the Large Messaging sample to best use the available memory.

- If you set Record detection to Delimited, use Delimiter to specify the delimiter to be used. Choose from the following options:
  - DOS or UNIX Line End, on UNIX systems, specifies the line feed character (<LF>, X'0A'), and, on Windows systems, specifies a carriage return character followed by a line feed character (<CR><LF>, X'0D0A'). The node treats both of these strings as delimiters, irrespective of the system on which the broker is running. If both strings can be seen in the same record, the node recognizes both as delimiters. The node does not recognize X'15' which, on z/OS systems, is the 'newline' byte; set this property to Custom Delimiter and set Custom delimiter to 15 if your input file is coded using EBCDIC new lines.
  - Custom Delimiter (hexadecimal), permits a sequence of bytes to be specified in Custom delimiter (hexadecimal)
- In Custom delimiter (hexadecimal), specify the delimiter byte or bytes to be used when Delimiter is set to Custom delimiter (hexadecimal). Specify this value as an even-numbered string of hexadecimal digits. The default is X'0A' and the maximum length of the string is 16 bytes (represented by 32 hexadecimal digits).
- If you set Record detection to Delimited, use Delimiter type to specify the type of delimiter. Permitted values are:
  - Infix. If you select this value, each delimiter separates records. If the data ends with a delimiter, the (zero length) data following the final delimiter is still propagated, although it contains no data.
  - Postfix. If you specify this value, each delimiter terminates records. If the data ends with a delimiter, no empty record is propagated after the delimiter. If the data does not end with a delimiter, it is processed as if a delimiter follows the final bytes of the data. Postfix is the default value.
- The TCPIPClientReceive node considers each occurrence of the delimiter in the input as either separating (infix) or terminating (postfix) each record. If the data begins with a delimiter, the node treats the (zero length) contents preceding that delimiter as a record and propagates an empty record to the flow. The delimiter is never included in the propagated message.

9. Use the **Validation** tab to provide validation based on the message set for predefined messages. For more information about validation, see "Validating messages" on page 204. For information about how to complete this tab, see "Validation tab properties" on page 1446.

Terminals and properties

The terminals of the TCPIPClientReceive node are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Out	The output terminal to which the message is routed if it is successfully retrieved from an external resource. If no errors occur in the input node, a message received from an external resource is always sent to the Out terminal first.
Timeout	The terminal to which a message is sent when the time specified in the Timeout waiting for a data record property has been exceeded. The message text is `Timeout value is exceeded`.

Terminal	Description
Failure	The output terminal to which the message is routed if an error occurs. These errors include failures caused by retry processing. Even if the Validation property is set, messages propagated to this terminal are not validated.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the TCPIPClientReceive node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	TCPIPClientReceive	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the TCPIPClientReceive node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Connection details	Yes	Yes		A string containing either the host name and port number to be used, or the name of a configurable service.	connectionDetails
Timeout waiting for a data record (seconds)	Yes	Yes	60	Specifies how long the node listens on a connection for more data after the first byte of data has arrived. You can specify any length of time in seconds.	timeoutWaitingForData

The Advanced properties of the TCPIPClientReceive node are described in the following table.

Property	M	C	Default	Description
Close connection	Yes	No	No	Controls when the connection is closed, or if it remains open. Valid options are: • No • After Timeout • After Data has been Received
Close input stream after a record has been received	Yes	No	Cleared	Specifies whether to close the input stream as soon as the data has been retrieved. When the connection input stream is reserved, no other node can use it without knowing the ID. By default, this property is not selected.

Property	M	C	Default	Description
Input Stream Modification	No	No	Leave unchanged	Specifies whether to reserve the input stream for use only by input and receive nodes that specify the connection ID, or to release the input stream at the end of the flow. Valid options are: • Leave unchanged • Release input stream • Reserve input stream (for use by future TCPIP nodes) • Reserve input stream (for use by future TCPIP nodes) then release at end of flow When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID. If the input stream is released after the message has been propagated, it is returned to the pool and becomes available for use by any input or receive node.
Output Stream Modification	No	No	Leave unchanged	Specifies whether to reserve this output stream or release it and return it to the pool for use by any output node. Valid options are: • Leave unchanged • Release output stream • Reserve output stream (for use by future TCPIP nodes) • Reserve output stream (for use by future TCPIP nodes) then release at end of flow

The Request properties of the TCPIPClientReceive node are described in the following table:

Property	M	C	Default	Description
Hostname location	Yes	No	$LocalEnvironment/TCPIP/Receive/Hostname	The message element location that contains the host name.
Port location	Yes	No	$LocalEnvironment/TCPIP/Receive/Port	The message element location that contains the port.
ID location	Yes	No	$LocalEnvironment//TCPIP/Receive/Id	The message element location that contains the ID.
Reply ID location	Yes	No	$LocalEnvironment/TCPIP/Receive/ReplyId	The message element location that contains the Reply ID.

The Result properties of the TCPIPClientReceive node are described in the following table:

Property	M	C	Default	Description
Output data location	No	No	$OutputRoot	The start location in the output message tree where the parsed elements from the bit string of the message are stored.
Copy local environment	No	No	Selected	Specifies if the local environment is copied to the output message.

The Input Message Parsing properties of the TCPIPClientReceive node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Message domain	No	No		The domain that is used to parse the incoming message.	
Message set	No	No		The name or identifier of the message set in which the incoming message is defined. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the Message Set property, or restore the reference to this message set project.	
Message type	No	No		The name of the incoming message.	
Message format	No	No		The name of the physical format of the incoming message.	
Message coded character set ID	Yes	No	Broker System Default	The ID of the coded character set that is used to interpret the data being read.	messageCodedCharSetIdProperty
Message encoding	Yes	No	Broker System Determined	The encoding scheme for numbers and large characters used to interpret the data that is being read. Valid values are Broker System Determined or a numeric encoding value. For more information about encoding, see "Converting data with message flows" on page 165.	messageEncodingProperty

The Parser Options properties of the TCPIPClientReceive node are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	On Demand	This property controls when an input message is parsed. Valid values are: • On Demand • Immediate • Complete For a full description of this property, see "Parsing on demand" on page 1449.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the syntax elements in the message tree have data types taken from the XML Schema.
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS domain. If you set this property, the message data appears under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or Input Message Parsing property, Message Domain, is XMLNS.

Property	M	C	Default	Description
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in an input message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in an input message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in an input message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the input message that are to be parsed opaquely by the XMLNSC parser.

The Records and Elements properties of the TCPIPClientReceive node are described in the following table:

Property	M	C	Default	Description
Record detection	Yes	No	Connection Closed	The mechanism used to identify records in the input data. Valid options are: • Connection Closed • Fixed Length • Delimited • Parsed Record Sequence
Length (bytes)	Yes	No	0	The length of each record, in bytes, when Fixed Length record detection is selected.
Delimiter	Yes	No	DOS or UNIX Line End	The type of delimiter bytes that separate, or ends, each record when Delimited record detection is selected. Valid options are: • DOS or UNIX Line End • Custom Delimiter (Hexadecimal)
Custom delimiter (hexadecimal)	No	No		The delimiter bytes, expressed in hexadecimal, when Delimited record detection and Custom Delimiter (Hexadecimal) are selected. This property is mandatory only if the Delimiter property is set to Custom Delimiter (Hexadecimal).
Delimiter type	Yes	No	Postfix	The location of the delimiter when Delimited record detection and Custom Delimiter (Hexadecimal) are selected. Valid options are: • Infix • Postfix This property is ignored unless the Delimiter property is set to Custom Delimiter (Hexadecimal).

The Validation properties of the TCPIPClientReceive node are described in the following table.

For a full description of these properties, see "Validation properties" on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are • None • Content and Value • Content • inherit	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are: • User Trace • Local Error Log • Exception • Exception List	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

TCPIPServerInput node

Use the TCPIPServerInput node to create a server connection to a raw TCPIP socket, and to receive data over that connection.

This topic contains the following sections:

Purpose

The TCPIPServerInput node listens on a port and, when a client socket connects to the port, the server socket creates a new connection for the client. Unlike the TCPIPClientInput node, the TCPIPServerInput node does not attempt to make a minimum number of connections, because the server end of the socket cannot initiate new connections, it can only accept them. The TCPIPServerInput node accepts connections up to a maximum value, which is specified in the MaximumConnections property of the TCPIPServer configurable service. By default, the broker can accept up to 100 server connections. For more information, see mqsicreateconfigurableservice command and mqsireportproperties command.

The first record of data is detected in accordance with properties on the node and then sent to the Out terminal. If an error occurs, including a timeout waiting for

data or the closure of a connection while waiting for the full record, the data is sent to the Failure terminal. If the connection closes and no data exists, a message is sent to the Close terminal. Although the message has no data, the local environment does have details of the connection that closed.

For both data and close events, the following local environment is created.

Table 19. Location in local environment

Location in local environment	Description
$LocalEnvironment/TCPIP/Input/ConnectionDetails/Type	The server.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/Hostname	The host name used to make a connection.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/Port	The port number used to make a connection.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/OpenTimestamp	The time stamp when the connection was first opened.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/CloseTimestamp	The time stamp when the connection was closed (null if not yet closed).
$LocalEnvironment/TCPIP/Input/ConnectionDetails/SequenceNumber/InputRecord	The sequence number of the message received on this connection. The first record has a sequencing number of 1; the second record has a sequencing number of 2; and so on.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/SequenceNumber/OutputRecord	The sequence number of the message sent on this connection. The first record has a sequencing number of 1; the second record has a sequencing number of 2; and so on.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/Id	The ID of the socket being used. This ID is an internal identifier used by WebSphere Message Broker to uniquely identify a connection.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/ReplyId	The Reply ID that is stored on this connection. It can be any text string.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/ClientDetails/Hostname	The fully qualified domain name of the computer from which the client connected.
$LocalEnvironment/TCPIP/Input/ConnectionDetails/ClientDetails/Address	The IP address of the computer from which the client connected.

The TCPIPServerInput node is contained in the **TCPIP** drawer of the palette, and is represented in the workbench by the following icon:

Message structure

The TCPIPServerInput node handles messages in the following message domains:

- MRM
- XMLNSC
- DataObject
- XMLNS
- JMSMap
- JMSStream

- MIME
- BLOB
- XML (this domain is deprecated; use XMLNSC)
- IDOC (this domain is deprecated; use MRM)

Using the TCPIPServerInput node in a message flow

Look at the following samples to see how to use the TCPIPServerInput node:

- TCPIP Client Nodes
- TCPIP Handshake

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

Configuring the TCPIPServerInput node

When you have put an instance of the TCPIPServerInput node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view.

All mandatory properties that do not have a default value defined are marked with an asterisk.

Configure the TCPIPServerInput node:

1. Optional: On the **Description** tab, enter a Short description, a Long description, or both. You can also rename the node on this tab.
2. On the **Basic** tab, set the properties that determine how the TCPIP connection is controlled.
   - Use the Connection details property to specify the port number to be used, or the name of a configurable service. This property is mandatory. The following formats are supported:
     - Configurable service name. This value is used to look up the port number in configurable services. For example, `TCPIPProfile1`.
     - <Port>. This value is the port number. For example, 1111
     - <Port>. This value is the port number. In this case, the host name is assumed to be localhost.
   - Use the Timeout waiting for a data record (seconds) property to specify how long the node listens on a connection for more data after the first byte of data has arrived. You can specify any length of time in seconds. The default is 60 seconds. When the specified time has been exceeded, all available data is sent to the Failure terminal.
3. On the **Advanced** tab, set the properties that determine how the data stream is controlled.
   - Use the Close connection property to specify when and how to close the connection.
     - Select No to leave the connection open. This value is the default.
     - Select After timeout to close the connection when a timeout occurs.
     - Select After data has been received to close the connection when the end of the record is found.
     - Select At end of flow to close the connection after the flow has been run.

- Select Close input stream after a record has been received to close the input stream as soon as the data has been retrieved. When the connection input stream is reserved, no other node can use it without specifying the ID. This property is not selected by default.
- Use the Input Stream Modification property to specify whether to reserve the input stream for use only by input and receive nodes that specify the connection ID, and, if reserved, whether to release the input stream at the end of the flow. These options are available only if you have not selected the Close input stream after a record has been received property.
  - Select Leave unchanged to leave the input stream as it was when it entered the node. This value is selected by default.
  - Select Reserve input stream (for use by future TCPIP input and receive nodes) to specify that this input stream can be used only by this node and by other receive nodes that request it by specifying the connection ID. When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID.
  - Select Reserve input stream (for use by future TCPIP input and receive nodes) then release at end of flow to specify that this input stream can be used only by this node and receive nodes that request it by specifying the connection ID. After the flow has been run, this input stream is returned to the pool and becomes available for use by any input or receive node.
- Use the Output Stream Modification property to specify whether to release the output stream.
  - Select Leave unchanged to leave the output stream as it was when it entered the node. This value is selected by default.
  - Select Release output stream and reset ReplyID to specify that this output stream is returned to the pool and is available for use by any output node. The ReplyID is passed in the local environment when leaving this node, but is reset for the next record on this connection.

4. On the **Input Message Parsing** tab, set values for the properties that the node uses to determine how to parse the incoming message.

   If the incoming message has an MQRFH2 header, you do not need to set values for the Input Message Parsing properties because the values are derived from the <mcd> folder in the MQRFH2 header; for example:

   ```
   <mcd><Msd>MRM</Msd><Set>DHM4UO906S001</Set><Type>receiptmsg1</Type>
   <Fmt>XML</Fmt></mcd>
   ```

   If you set values, and those values differ from those in the MQRFH2 header, the values in the MQRFH2 header take precedence.

   - In Message domain, select the name of the parser that you are using from the list. The default is BLOB. You can choose from the following options:
     - MRM
     - XMLNSC
     - DataObject
     - XMLNS
     - JMSMap
     - JMSStream
     - MIME
     - BLOB
     - XML (this domain is deprecated; use XMLNSC)
     - IDOC (this domain is deprecated; use MRM)

     You can also specify a user-defined parser, if appropriate.

- If you are using the MRM or IDOC parser, or the XMLNSC parser in validating mode, select the Message set that you want to use. The list contains the message sets that are available when you select MRM, XMLNSC, or IDOC as the domain.
- If you are using the MRM parser, select the correct message type from the list in Message type. This list is populated with available message types when you select the MRM parser.
- If you are using the MRM or IDOC parser, select the correct message format from the list in Message format. This list is populated with available message formats when you select the MRM or IDOC parser.
- Specify the message coded character set ID in Message coded character set ID.
- Select the message encoding from the list in Message encoding or specify a numeric encoding value. For more information about encoding, see "Converting data with message flows" on page 165.

5. On the **Parser Options** sub-tab:
   a. Parse timing is, by default, set to On Demand, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see "Parsing on demand" on page 1449.
   b. If you are using the XMLNSC parser, set values for the properties that determine how the XMLNSC parser operates. For more information, see "Manipulating messages in the XMLNSC domain" on page 408.
6. Use the **Retry** tab to define how retry processing is performed when a flow fails. You can set the following properties:
   - Retry mechanism determines the action that occurs should the flow fail. Choose from the following values:
     – Select Failure for the node to report a failure without any retry attempts.
     – Select Short retry for the node to retry before reporting a failure if the condition persists. The number of times that it retries is specified in Retry threshold.
     – Select Short retry and long retry for the node to retry, first using the value in Retry threshold as the number of attempts it should make. If the condition persists after the Retry threshold has been reached, the node then uses the Long retry interval between attempts.
   - Specify the Retry threshold:The number of times the node retries the flow transaction if the Retry mechanism property is set to either Short retry or Short retry and long retry.
   - Specify the Short retry interval:The length of time, in seconds, to wait between short retry attempts.
   - Specify the Long retry interval:The length of time to wait between long retry attempts until a message is successful, the message flow is stopped, or the message flow is redeployed. The broker property **MinLongRetryInterval** defines the minimum value that the Long retry interval can take. If the value is lower than the minimum, the broker value is used.
7. Use the **Records and Elements** tab to specify how the data is interpreted as records.
   - Use the Record detection property to determine how the data is split into records, each of which generates a single message. Choose from the following options:

- End of stream specifies that all of the data sent in the data stream is a single record.
- Fixed Length specifies that each record is a fixed number of bytes in length. Each record contains the number of bytes specified in the Length property.
- Select Delimited if the records that you are processing are separated, or ended, by a DOS or UNIX line end or by a sequence of user-defined delimiter bytes. Specify the delimiter and delimiter type in the Delimiter and Delimiter type properties.
- Select Parsed Record Sequence if the data contains a sequence of one or more records that are serially recognized by the parser specified in Message domain. The node propagates each recognized record as a separate message. If you select the Record detection option, the parser specified in Message domain must be either XMLNSC or MRM (either CWF or TDS physical format).

- If you set Record detection to Fixed Length, use Length to specify the required length of the output record. This value must be between 1 byte and 100 MB. The default is 80 bytes.

  If you set Record detection to End of stream, Fixed Length, or Delimited, a limit of 100 MB applies to the length of the records. If you set Record detection toParsed Record Sequence, the TCPIPServerInput node does not determine or limit the length of a record. Nodes that are downstream in the message flow might try to determine the record length or process a very long record. If you intend to process large records in this way, ensure that your broker has sufficient memory. You might need to apply flow techniques described in the Large Messaging sample to best use the available memory.
- If you set Record detection to Delimited, use Delimiter to specify the delimiter to be used. Choose from the following values:
  - DOS or UNIX Line End, which, on UNIX systems, specifies the line feed character (<LF>, X'0A'), and, on Windows systems, specifies a carriage return character followed by a line feed character (<CR><LF>, X'0D0A'). The node treats both of these strings as delimiters, irrespective of the system on which the broker is running. If both strings are seen in the same record, the node recognizes both as delimiters. The node does not recognize X'15', which, on z/OS systems, is the 'newline' byte; specify a value of Custom Delimiter in this property and a value of 15 in the Custom delimiter property if your input data is coded using EBCDIC new lines.
  - Custom Delimiter, which permits a sequence of bytes to be specified in Custom delimiter
- In Custom delimiter, specify the delimiter byte or bytes to be used when Delimiter is set to Custom delimiter. Specify this value as an even-numbered string of hexadecimal digits. The default is X'0A' and the maximum length of the string is 16 bytes (represented by 32 hexadecimal digits).
- If you set Record detection to Delimited, use Delimiter type to specify the type of delimiter. Permitted values are:
  - Infix. If you select this value, each delimiter separates records. If the data ends with a delimiter, the (zero length) data that follows the final delimiter is still propagated although it contains no data.
  - Postfix. If you specify this value, each delimiter ends records. If the data ends with a delimiter, no empty record is propagated after the delimiter.

If the data does not end with a delimiter, it is processed as if a delimiter follows the final bytes of the data. Postfix is the default value.

- The TCPIPServerInput node considers each occurrence of the delimiter in the input as either separating (infix) or terminating (postfix) each record. If the data begins with a delimiter, the node treats the (zero length) contents preceding that delimiter as a record and propagates an empty record to the flow. The delimiter is never included in the propagated message.

8. Use the **Validation** tab to provide validation based on the message set for predefined messages. For more information about validation, see "Validating messages" on page 204. For information about how to provide validation for this tab, see "Validation tab properties" on page 1446.
9. On the **Transactions** tab, set the transaction mode. Although TCPIP operations are non-transactional, the transaction mode on this input node determines whether the rest of the nodes in the flow are to be executed under point of consistency or not. Select Yes if you want the flow updates to be treated transactionally (if possible) or No if you do not. The default for this property is No.
10. Optional: On the **Instances** tab, set values for the properties that determine the additional instances (threads) that are available for a node. For more details, see "Configurable message flow properties" on page 1324.

Terminals and properties

The terminals of the TCPIPServerInput node are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is routed if an error occurs. These errors include failures caused by retry processing. Even if the Validation property is set, messages propagated to this terminal are not validated.
Out	The output terminal to which the message is routed if it is successfully retrieved from an external resource. If no errors occur within the input node, a message that is received from an external resource is always sent to the Out terminal first.
Close	The output terminal to which the message is routed if the connection closes.
Catch	The output terminal to which the message is routed if an exception is thrown downstream and caught by this node. Exceptions are caught only if this terminal is attached.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the TCPIPServerInput node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	TCPIPServerInput	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the TCPIPServerInput node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Connection details	Yes	Yes		A string containing the port number to be used, or the name of a configurable service.	connectionDetails
Timeout waiting for a data record (seconds)	Yes	Yes	60	Specifies how long the node listens on a connection for more data after the first byte of data has arrived. You can specify any length of time in seconds.	timeoutWaitingForData

The Advanced properties of the TCPIPServerInput node are described in the following table.

Property	M	C	Default	Description
Close connection	Yes	No	No	Controls when the connection is closed, or if it remains open. Valid options are: • No • After Timeout • After Data has been Received • At End of Flow
Close input stream after a record has been received	Yes	No	Cleared	Specifies whether to close the input stream as soon as the data has been retrieved. When the connection input stream is reserved, no other node can use it without knowing the ID. By default, this option is not selected.
Input Stream Modification	No	No	Leave unchanged	Specifies whether to reserve the input stream for use only by input and receive nodes that specify the connection ID, and, if reserved, whether to release the input stream at the end of the flow. Valid options are: • Leave unchanged • Reserve input stream (for use by future TCPIP nodes) • Reserve input stream (for use by future TCPIP nodes) then release at end of flow When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID. If the input stream is released at the end of the flow, it is returned to the pool and becomes available for use by any input or receive node.
Output stream modification	No	No	Leave unchanged	Specifies whether this output stream is released and returned to the pool for use by any output node. Valid options are: • Leave unchanged • Release output stream and reset ReplyID If you select Release output stream and reset ReplyID, the ReplyID is passed in the local environment when leaving this node, but is reset for the next record on this connection.

The Input Message Parsing properties of the TCPIPServerInput node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Message domain	No	No		The domain that is used to parse the incoming message.	
Message set	No	No		The name or identifier of the message set in which the incoming message is defined. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the Message Set property, or restore the reference to this message set project.	
Message type	No	No		The name of the incoming message.	
Message format	No	No		The name of the physical format of the incoming message.	
Message coded character set ID	Yes	No	Broker System Default	The ID of the coded character set used to interpret the data being read.	messageCodedCharSetIdProperty
Message encoding	Yes	No	Broker System Determined	The encoding scheme for numbers and large characters used to interpret the data being read. Valid values are Broker System Determined or a numeric encoding value. For more information about encoding, see "Converting data with message flows" on page 165.	messageEncodingProperty

The Parser Options properties of the TCPIPServerInput node are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	On Demand	This property controls when an input message is parsed. Valid values are: • On Demand • Immediate • Complete For a full description of this property, see "Parsing on demand" on page 1449.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the syntax elements in the message tree have data types taken from the XML Schema.
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the message data appears under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or Input Message Parsing property, Message Domain, is XMLNS.

Property	M	C	Default	Description
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in an input message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in an input message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in an input message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the input message that are to be opaquely parsed by the XMLNSC parser.

The Records and Elements properties of the TCPIPServerInput node are described in the following table:

Property	M	C	Default	Description
Record detection	Yes	No	End of Stream	The mechanism used to identify records in the input data. Valid options are: • End of Stream • Fixed Length • Delimited • Parsed Record Sequence
Length (bytes)	Yes	No	0	The length of each record, in bytes, when Fixed Length record detection is selected.
Delimiter	Yes	No	DOS or UNIX Line End	The type of delimiter bytes that separates, or ends, each record when Delimited record detection is selected. Valid options are: • DOS or UNIX Line End • Custom Delimiter (Hexadecimal)
Custom delimiter (hexadecimal)	No	No		The delimiter bytes, expressed in hexadecimal, when Delimited record detection and Custom Delimiter (Hexadecimal) are selected. This property is mandatory only if the Delimiter property is set to Custom Delimiter (Hexadecimal).
Delimiter type	Yes	No	Postfix	The location of the delimiter when Delimited record detection and Custom Delimiter (Hexadecimal) are selected. Valid options are: • Infix • Postfix This property is ignored unless the Delimiter property is set to Custom Delimiter (Hexadecimal).

The Retry properties of the TCPIPServerInput node are described in the following table:

Property	M	C	Default	Description	mqsiapplybaroverride command property
Retry mechanism	Yes	No	Failure	How the node handles a flow failure. Valid options are: • Failure • Short Retry • Short and Long Retry	
Retry threshold	Yes	Yes	0	The number of times to retry the flow transaction when Retry mechanism is Short retry.	retryThreshold
Short retry interval (seconds)	No	Yes	0	The interval, in seconds, between each retry if Retry threshold is not zero.	shortRetryThreshold
Long retry interval (seconds)	No	Yes	300	The interval between retries if Retry mechanism is Short and Long Retry and the retry threshold has been exhausted.	longRetryThreshold

The Validation properties of the TCPIPServerInput node are described in the following table.

For a full description of these properties, see "Validation properties" on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are • None • Content and Value • Content	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. Valid values are: • User Trace • Local Error Log • Exception • Exception List	

The Transactions properties of the TCPIPServerInput node are described in the following table:

Property	M	C	Default	Description
Transaction mode	No	Yes	No	The transaction mode on this input node determines whether the rest of the nodes in the flow are executed under point of consistency. Valid options are: • Yes • No

The Instances properties of the TCPIPServerInput node are described in the following table. For a full description of these properties, see "Configurable message flow properties" on page 1324.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Additional instances pool	No	Yes	Use Pool Associated with Message Flow	The pool from which additional instances are obtained. • If you select Use Pool Associated with Message Flow, additional instances are obtained from the message flow pool. • If you select Use Pool Associated with Node, additional instances are allocated from the node's additional instances based on the number specified in the Additional instances property.	componentLevel
Additional instances	No	Yes	0	The number of additional instances that the node can start if the Additional instances pool property is set to Use Pool Associated with Node.	additionalInstances

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

TCPIPServerOutput node

Use the TCPIPServerOutput node to create a server connection to a raw TCP/IP socket, and to send data over the connection to an external application.

This topic contains the following sections:

Purpose

The TCPIPServerOutput listens on a TCP/IP port and waits for a client node to make connection with the port. When the client node connects to the port, the server node creates a connection for the client. The connections are not made directly by the node but are obtained from a connection pool managed by the WebSphere Message Broker execution group.

The execution group uses the default TCPIPServer configurable service to determine which attributes are used for the socket connection. However, if the configurable service is set on the node, the configurable service is used for all the properties, including the host and port number.

When the connection has been established, the data is sent. If the data is not sent successfully within the time limit specified by the node's Timeout sending a data record property, an exception is thrown.

Properties in the local environment can override the TCP/IP connection used by the node.

Table 20. Input local environment properties

Location in local environment	Description
$LocalEnvironment/Destination/TCPIP/Output/Hostname	The host name used to make a connection.
$LocalEnvironment/Destination/TCPIP/Output/Port	The port number used to make a connection.
$LocalEnvironment/Destination/TCPIP/Output/Id	The ID of the socket being used. This ID is an internal identifier used by WebSphere Message Broker to uniquely identify a connection.
$LocalEnvironment/Destination/TCPIP/Output/ReplyId	The Reply ID that is stored on this connection. It can be any text string.

You can dynamically choose the connection details (host name and port number), and the connection used (ID), using these properties. The Reply ID can also be set on the connection, which enables a string to be stored in the connection and to be displayed in the local environment. This behavior can be used to store Reply IDs from other TCPIP nodes or from other transports such WebSphere MQ.

The output of the node contains the same information as the input, plus any fields that are missing from the input are updated with details from the connection used. For example, if the Id property is not provided as input (because you want to create a connection or reuse a pool connection), the output local environment contains the ID of the connection that is used.

Table 21. Output local environment properties

Location in local environment	Description
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/Hostname	The host name used to make a connection.
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/Port	The port number used to make a connection.
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/OpenTimestamp	The time stamp when the connection was first opened.
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/CloseTimestamp	The time stamp when the connection was closed (null if not yet closed).
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/SequenceNumber	The sequence number of the message received on this connection. The first record has a sequencing number 1; the second record 2; and so on.
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/Id	The ID of the socket being used. This ID is an internal identifier used by WebSphere Message Broker to uniquely identify a connection.
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/ReplyId	The Reply ID that is stored on this connection. It can be any text string.
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/ClientDetails/Hostname	The fully qualified domain name of the computer from which the client connected.
$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails/ClientDetails/Address	The IP address of the computer from which the client connected.

If the connection closes (or any other type of exception occurs) while using the TCP/IP transport, an exception is thrown. This exception goes to the Failure terminal if it is connected; otherwise the exception goes back down the flow.

The node also has a Close input terminal. If a message is sent to this terminal, the connection is closed using a combination of the details provided in the node and the local environment.

The TCPIPServerOutput node is contained in the **TCPIP** drawer of the palette and is represented in the workbench by the following icon:

Using the TCPIPServerOutput node in a message flow

You can use the TCPIPServerOutput node in any message flow that needs to send data to an external application. Look at the following samples to see how to use it:

- TCPIP Client Nodes
- TCPIP Handshake

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

Configuring the TCPIPServerOutput node

When you have put an instance of the TCPIPServerOutput node into a message flow, you can configure it (for more information, see "Configuring a message flow node" on page 276). The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk in that view.

To configure the TCPIPServerOutput node:

1. Optional: On the **Description** tab, enter a short description, a long description, or both. You can also rename the node on this tab.
2. On the **Basic** tab, set the properties that determine how the TCP/IP connection is controlled.
   - Use the Connection details property to specify the port number to be used, or the name of a configurable service. This property is mandatory. The following formats are supported:
     - Configurable service name. This value is used to look up the port number in configurable services; for example, `TCPIPProfile1`.
     - <Port>. This value is the port number; for example, `1111`
     - <Port>. This value is the port number. In this case the host name is assumed to be localhost.
   - Use the Timeout sending a data record (seconds) property to specify how long the node waits when trying to send data. You can specify any length of time in seconds. When the specified time has been exceeded, all available data is sent to the Failure terminal. The default is 60 seconds.
3. On the **Advanced** tab, set the properties that determine how the data stream is controlled.
   - Use the Send to property to specify whether the data is to be sent to one connection or to all available connections.

- Select One connection to send the message to only one connection, as specified by the node properties and message overrides. This value is the default.
- Select All available connections to send the data to all available connections.

- Use the Close connection property to specify when and how to close the connection.
  - Select No to leave the connection open. This value is the default.
  - Select After timeout to close the connection when a timeout occurs.
  - Select After data has been sent to close the connection when the end of the record has been sent.
- Select Close output stream after a record has been sent to close the output stream as soon as the data has been sent. By default, this property is not selected.
- Use the Output Stream Modification property to specify whether to reserve or release the output stream. These options are available only if you have not selected the Close output stream after a record has been sent property.
  - Select Leave unchanged to leave the output stream as it was when it entered the node. This value is selected by default.
  - Select Release output stream to specify that this output stream is returned to the pool and is available for use by any output node.
  - Select Reserve output stream (for use by future TCPIP output nodes) to specify that this output stream can be used only by this node and by other output nodes that request it by specifying the connection ID. When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID.
  - Select Reserve output stream (for use by future TCPIP output nodes) then release after propagate to specify that this output stream can be used only by this node and output nodes that request it by specifying the correct connection ID. After the message has been propagated, this output stream is returned to the pool and becomes available for use by any output node.
- Use the Input Stream Modification property to reserve the input stream for use only by input and receive nodes that specify the connection ID, or to release the input stream at the end of the message flow.
  - Select Leave unchanged to leave the input stream as it was when it entered the node. This value is selected by default.
  - Select Release input stream to specify that this input stream is returned to the pool and is available for use by any input or receive node.
  - Select Reserve input stream (for use by future TCPIP input and receive nodes) to specify that this input stream can be used only by this node and by other input or receive nodes that request it by specifying the connection ID. When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID.
  - Select Reserve input stream (for use by future TCPIP input and receive nodes) then release after propagate to specify that this input stream can be used only by this node and receive nodes that request it by specifying the correct connection ID. After the message has been propagated, this input stream is returned to the pool and becomes available for use by any input or receive node.

4. On the **Request** tab, specify the location of the data to be written. You can specify the properties on this tab as XPath or ESQL expressions. Content Assist

is available in the Properties view and also in the XPath Expression Builder, which you can call by using the **Edit** button to the right of each property.

a. In Data location, specify the input data location, which is the location in the input message tree that contains the record to be written. The default value is $Body, which is the entire message body ($InputRoot.Body).

   When you are specifying this property, and the data in the message tree that it identifies is owned by a model-driven parser, such as the MRM parser or XMLNSC parser, be aware of the following considerations:

   - If you are using MRM CWF format, ensure that the identified message tree exists as a message definition. If this message tree is defined as a global element only, exceptions BIP5180 and BIP5167 are generated.
   - If you are using MRM TDS format, the serialization of the identified message is successful if the element is defined as a global element or message. However, if the identified field is not found as a global element or message, note that:
     – If this field is a leaf field in the message tree, the field is written as self-defining. No validation occurs even if validation is enabled.
     – If this field is a complex element, an internal exception is generated, BIP5522, indicating that the logical type cannot be converted to a string.
   - If you are using MRM XML, the events are similar to the MRM TDS format except that, if the field is a complex element, it is written as self-defining.
   - If you use the XMLNSC parser, no validation occurs, even if validation is enabled.

b. In Port location, specify the location of the value to override the Port that is set in the Connection details property of the **Basic** tab. If you do not specify a location, the default value is $LocalEnvironment/Destination/TCPIP/Output/Port.

c. In ID location, specify the location of the Id of the socket being used. This internal identifier is used by WebSphere Message Broker to uniquely identify a connection. If you do not specify a location, the default value is $LocalEnvironment/Destination/TCPIP/Output/Id.

d. In Reply ID location, specify the location of the Reply ID that is stored on the connection that is being used. The Reply ID can be used when data is returned in an input node. If you do not specify a location, the default value is $LocalEnvironment/Destination/TCPIP/Output/ReplyId.

5. Use the **Records and Elements** tab to specify how the TCPIPServerOutput node writes the record that is derived from the message.

   - In Record definition, choose from the following values:
     – Record is Unmodified Data specifies that records are left unchanged. This value is the default.
     – Record is Fixed Length Data specifies that records are padded to a specified length if necessary. You specify this length in the Length property. If the record is longer than the value specified in Length, the node generates an exception. Use the Padding byte property to specify the byte to be used for padding the message to the required length.
     – Record is Delimited Data specifies that records are separated by a delimiter and accumulated by concatenation. The delimiter is specified by the Delimiter, Custom delimiter, and Delimiter type properties. The file is finished only when a message is received on the Finish File terminal.

- In Length, specify the length (in bytes) of records when Record definition is set to Record is Fixed Length Data. Records longer than this value cause an exception to be thrown. This value must be between 1 byte and 100 MB. The default is 80 bytes.
- When Record definition is set to Record is Fixed Length Data, use Padding byte to specify the byte to be used when padding records to the specified length if they are shorter than this length. Specify this value as two hexadecimal digits. The default value is X'20'.
- In Delimiter, specify the delimiter to be used if you set Record definition to Record is Delimited Data. Choose from:
  - Broker System Line End specifies that a line end sequence of bytes is used as the delimiter as appropriate for the file system on which the broker is running. This value is the default. For example, on Windows systems, this line end is a 'carriage-return, line-feed' pair (X'0D0A'); on UNIX systems, it is a single 'line-feed' byte (X'0A'); on z/OS systems, it is a 'newline' byte (X'15').
  - Custom Delimiter specifies that the explicit delimiter sequence defined in the Custom delimiter property is to be used to delimit records.
- In Custom delimiter, specify the delimiter sequence of bytes to be used to delimit records when Delimiter is set to Custom Delimiter. Specify this value as an even-numbered string of hexadecimal digits. The default is X'0A' and the maximum length of the string is 16 bytes.
- If you set Record definition to Record is Delimited Data, use Delimiter type to specify how the delimiter is to separate records. Choose from the following values:
  - Postfix specifies that the delimiter is added after each record that is written. This value is the default.
  - Infix specifies that the delimiter is inserted between any two adjacent records only.

6. On the **Validation** tab, specify the parser validation properties of the node. For more information about validation, see "Validating messages" on page 204. For information about how to provide validation for this tab, see "Validation tab properties" on page 1446.

Terminals and properties

The TCPIPServerOutput node terminals are described in the following table.

Terminal	Type	Description
In	Input data	The input terminal that accepts a message for processing by the node.
Close	Input control	The input terminal to which a message is routed when the connection given in the local environment is closed.
Out	Output data	The output terminal to which the message is routed if it is successfully sent to an external resource. The message received on the In terminal is propagated to the Out terminal and is left unchanged except for the addition of status information.
Close	Output control	The output terminal to which a message propagated from the Close input terminal is routed.
Failure	Output data	The output terminal to which the message is routed if a failure is detected in the node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a

value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file for deployment).

The Description properties of the TCPIPServerOutput node are described in the following table:

Property	M	C	Default	Description
Node name	No	No	TCPIPServerOutput	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the TCPIPServerOutput node are described in the following table:

Property	M	C	Default	Description	mqsiapplybaroverride command property
Connection details	Yes	Yes		A string containing the port number to be used, or the name of a configurable service.	connectionDetails
Timeout sending a data record (seconds)	Yes	Yes	60	Specifies how long the node waits when trying to send data. You can specify any length of time in seconds.	timeoutSendingData

The Advanced properties of the TCPIPServerOutput node are described in the following table.

Property	M	C	Default	Description
Close connection	Yes	No	No	Controls when the connection is closed, or if it remains open. Valid options are: • No • After Timeout • After Data has been Sent
Close output stream after a record has been sent	Yes	No	Cleared	Specifies whether to close the output stream as soon as the data has been sent. By default, this property is not selected.
Output Stream Modification	No	No	Leave unchanged	Specifies whether to reserve this output stream or release it and return it to the pool for use by any output node. Valid options are: • Leave unchanged • Release output stream • Reserve output stream (for use by future TCPIP nodes) • Reserve output stream (for use by future TCPIP nodes) then release at end of flow

Property	M	C	Default	Description
Input Stream Modification	No	No	Leave unchanged	Specifies whether to reserve the input stream for use only by input and receive nodes that specify the connection ID, or to release the input stream at the end of the flow. Valid options are: • Leave unchanged • Release input stream • Reserve input stream (for use by future TCPIP nodes) • Reserve input stream (for use by future TCPIP nodes) then release at end of flow When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID. If the input stream is released after the message has been propagated, it is returned to the pool and becomes available for use by any input or receive node.
Send to:	Yes	No	One connection	Specifies whether the data is to be sent to one connection or to all available connections. Valid options are: • One Connection • All Available Connections

The Request properties of the TCPIPServerOutput node are described in the following table:

Property	M	C	Default	Description
Data location	Yes	No	$Body	The location in the input message tree that contains the record to be written.
Port location	Yes	No	$LocalEnvironment/Destination/TCPIP/Output/Port	The message element location that contains the port.
ID	Yes	No	$LocalEnvironment/Destination/TCPIP/Output/Id	The message element location that contains the ID.
Reply ID location	Yes	No	$LocalEnvironment/Destination/TCPIP/Output/ReplyId	The message element location that contains the Reply ID.

The Records and Elements properties of the TCPIPServerOutput node are described in the following table:

Property	M	C	Default	Description
Record definition	Yes	No	Record is Unmodified Data	This property controls how the records derived from the message are written. Valid options are: • Record is Unmodified Data • Record is Fixed Length Data • Record is Delimited Data
Length (bytes)	Yes	No	0	The required length of the output record. This property applies only when Record definition is set to Record is Fixed Length Data.
Padding byte (hexadecimal)	Yes	No	20	The two-digit hexadecimal byte to be used to pad short messages when Record definition is set to Record is Fixed Length Data.

Property	M	C	Default	Description
Delimiter	Yes	No	Broker System Line End	The delimiter to be used when Record definition is set to Record is Delimited Data. Valid options are: • Broker System Line End • Custom Delimiter (Hexadecimal)
Custom delimiter (hexadecimal)	No	No	None	The delimiter byte sequence to be used when Record definition is set to Record is Delimited Data and Delimiter is set to Custom Delimiter (Hexadecimal).
Delimiter type	Yes	No	Postfix	This property specifies the way in which the delimiters are inserted between records when Record definition is set to Record is Delimited Data. Valid options are: • Infix • Postfix

The Validation properties of the TCPIPServerOutput node are described in the following table.

For a full description of these properties, see "Validation properties" on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	Inherit	This property controls whether validation takes place. Valid values are: • None • Content and Value • Content • Inherit	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are: • User Trace • Local Error Log • Exception • Exception List	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

TCPIPServerReceive node

Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

This topic contains the following sections:

- "Configuring the TCPIPServerReceive node" on page 1267
- "Terminals and properties" on page 1271

Purpose

The TCPIPServerReceive node waits for data to be received on a TCP/IP connection, and retrieves the data. If the connection is closed, an exception is thrown.

When a connection is established, the data is sent to the TCPIPServerReceive node. If the TCPIPServerReceive node fails to receive all the data within the time specified in the Timeout waiting for a data record property, the message is sent to the Timeout terminal; if no Timeout terminal is connected, an exception is thrown.

Properties in the local environment can override the TCP/IP connection used by the node.

Table 22. Input local environment properties

Location in local environment for input to node	Description
$LocalEnvironment//TCPIP/Receive/Hostname	The host name used to make a connection.
$LocalEnvironment//TCPIP/Receive/Port	The port number used to make a connection.
$LocalEnvironment/TCPIP/Receive/Id	The ID of the socket being used. This ID is an internal identifier used by WebSphere Message Broker to uniquely identify a connection.
$LocalEnvironment/TCPIP/Receive/ReplyId	The Reply ID to be stored on this connection. This ID can then be used when data is returned on an input node. The Reply ID can be any text string.

These properties enable the connection details (host name and port number) and the connection used (ID) to be chosen dynamically. The Reply ID can also be set on the connection, which enables a string to be stored in the connection and to be displayed in the local environment. In this way, you can store Reply IDs from other TCPIP nodes or from other transports, such as WebSphere MQ.

When a record has been retrieved, the ConnectionDetails field in the local environment tree is populated with the details of the connection that is being used.

Table 23. Output local environment properties

Location in local environment for output from node	Description
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/Type	The Server.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/Hostname	The host name used to make a connection.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/Port	The port number used to make a connection.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/OpenTimestamp	The time stamp when the connection was first opened.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/CloseTimestamp	The time stamp when the connection was closed (null if not yet closed).
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/SequenceNumber/InputRecord	The sequence number of the message that is received on this connection. The first record has a sequencing number of 1; the second record is 2; and so on.

Table 23. Output local environment properties (continued)

Location in local environment for output from node	Description
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/SequenceNumber/OutputRecord	The sequence number of the message that is sent on this connection. The first record has a sequencing number of 1; the second record is 2; and so on.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/Id	The ID of the socket being used. This ID is an internal identifier used by WebSphere Message Broker to uniquely identify a connection.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/ReplyId	The Reply ID that is stored on this connection. It can be any text string.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/ClientDetails/Hostname	The fully qualified domain name of the computer from which the client connected.
$LocalEnvironment/TCPIP/Receive/ConnectionDetails/ClientDetails/Address	The IP address of the computer from which the client connected.

The TCPIPServerReceive node is contained in the **TCPIP** drawer of the palette, and is represented in the workbench by the following icon:

Message structure

The TCPIPServerReceive node handles messages in the following message domains:

- MRM
- XMLNSC
- DataObject
- XMLNS
- JMSMap
- JMSStream
- MIME
- BLOB
- XML (this domain is deprecated; use XMLNSC)
- IDOC (this domain is deprecated; use MRM)

Using the TCPIPServerReceive node in a message flow

Look at the following samples to see how to use the TCPIPServerReceive node:

- TCPIP Client Nodes
- TCPIP Handshake

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

Configuring the TCPIPServerReceive node

When you have put an instance of the TCPIPServerReceive node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view.

All mandatory properties that do not have a default value defined are marked with an asterisk.

Configure the TCPIPServerReceive node:

1. Optional: On the **Description** tab, enter a Short description, a Long description, or both. You can also rename the node on this tab.
2. On the **Basic** tab, set the properties that determine how the TCP/IP connection is controlled.
   - Use the Connection details property to specify either the host name and port number to be used, or the name of a configurable service. This property is mandatory. The following formats are supported:
     - Configurable service name. This value is used to look up the port and host name in configurable services. For example, `TCPIPProfile1`.
     - <Hostname>:<Port>. This value is the host name followed by the port number (separated by a colon); for example, `tcpip.server.com:1111`
     - <Port>. This value is the port number. In this case, the host name is assumed to be localhost.
   - Use the Timeout waiting for a data record (seconds) property to specify how long the node listens on a connection for more data after the first byte of data has arrived. You can specify any length of time in seconds. The default is 60 seconds. When the specified time has been exceeded, all available data is sent to the Failure terminal.
3. On the **Advanced** tab, set the properties that determine how the data stream is controlled.
   - Use the Close connection property to specify when and how to close the connection.
     - Select No to leave the connection open. This value is the default.
     - Select After timeout to close the connection when a timeout occurs.
     - Select After data has been received to close the connection when the end of the record is found.
   - Select Close input stream after a record has been received to close the input stream as soon as the data has been retrieved. By default this property is not selected. When the connection input stream is reserved, no other node can use it without knowing the ID.
   - Use the Input Stream Modification property to specify whether to reserve the input stream for use only by input and receive nodes that specify the connection ID, or to release the input stream at the end of the flow.
     - Select Leave unchanged to leave the input stream as it was when it entered the node. This value is selected by default.
     - Select Release input stream to specify that this input stream is returned to the pool and is available for use by any input or receive node.
     - Select Reserve input stream (for use by future TCPIP input and receive nodes) to specify that this input stream can be used only by this node and by other input or receive nodes that request it by specifying the connection ID. When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID.

- Select Reserve input stream (for use by future TCPIP input and receive nodes) then release after propagate to specify that this input stream can be used only by this node and receive nodes that request it by specifying the correct connection ID. After the message has been propagated, this input stream is returned to the pool and becomes available for use by any input or receive node.

- Use the Output Stream Modification property to specify whether to reserve or release the output stream. These options are available only if you have not selected the Close output stream after a record has been sent property.
  - Select Leave unchanged to leave the output stream as it was when it entered the node. This value is selected by default.
  - Select Release output stream to specify that this output stream is returned to the pool and is available for use by any output node.
  - Select Reserve output stream (for use by future TCPIP output nodes) to specify that this output stream can be used only by this node and by other output nodes that request it by specifying the connection ID. When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID.
  - Select Reserve output stream (for use by future TCPIP output nodes) then release after propagate to specify that this output stream can be used only by this node and output nodes that request it by specifying the correct connection ID. After the message has been propagated, this output stream is returned to the pool and becomes available for use by any output node.

4. On the **Request** tab, specify the location of the data to be written. You can specify the properties on this tab as XPath or ESQL expressions. Content Assist is available in the Properties view and also in the XPath Expression Builder, which you can run by clicking **Edit** to the right of each property.
   - In Hostname location, specify the location of the value to override the Hostname that is set in the Connection details property of the **Basic** tab. If you do not specify a location, the default value is $LocalEnvironment/TCPIP/Receive/Hostname.
   - In Port location, specify the location of the value to override the Port that is set in the Connection details property of the **Basic** tab. If you do not specify a location, the default value is $LocalEnvironment/TCPIP/Receive/Port.
   - In ID location, specify the location of the Id of the socket being used. This internal identifier is used by WebSphere Message Broker to uniquely identify a connection. If you do not specify a location, the default value is $LocalEnvironment/TCPIP/Receive/Id.
   - In Reply ID location, specify the location of the Reply ID that is stored on the connection that is being used. The Reply ID can be used when data is returned in an input node. If you do not specify a location, the default value is $LocalEnvironment/TCPIP/Receive/ReplyId.
5. On the **Result** tab, set values for the properties that determine where the reply is stored.
   - Use the Output data location property to specify the start location in the output message tree where the parsed elements from the bit string of the message are stored. The default value is `$OutputRoot`.
   - Use the Copy local environment property to specify whether the local environment is copied to the output message.
     - If Copy local environment is selected, a new copy of the local environment is created in the tree, and it is populated with the contents of the local environment from the preceding node. Therefore, if a node

changes the local environment, the upstream nodes are not affected by those changes because they have their own copies. This value is the default.

– If Copy local environment is not selected, the node does not generate its own copy of the local environment, but uses the local environment that is passed to it by the preceding node. Therefore, if a node changes the local environment, the changes are reflected by the upstream nodes.

6. On the **Input Message Parsing** tab, set values for the properties that the node uses to determine how to parse the incoming message.

   If the incoming message has an MQRFH2 header, you do not need to set values for the Input Message Parsing properties because the values are derived from the <mcd> folder in the MQRFH2 header; for example:

   ```
   <mcd><Msd>MRM</Msd><Set>DHM4UO906S001</Set><Type>receiptmsg1</Type>
   <Fmt>XML</Fmt></mcd>
   ```

   If you set values, and if they differ from the values in the MQRFH2 header, the values in the MQRFH2 header take precedence.

   - In Message domain, select the name of the parser that you are using from the list. The default is BLOB. You can choose from the following options:
     – MRM
     – XMLNSC
     – DataObject
     – XMLNS
     – JMSMap
     – JMSStream
     – MIME
     – BLOB
     – XML (this domain is deprecated; use XMLNSC)
     – IDOC (this domain is deprecated; use MRM)

     You can also specify a user-defined parser, if appropriate.
   - If you are using the MRM or IDOC parser, or the XMLNSC parser in validating mode, select the Message set that you want to use. The list contains the message sets that are available when you select MRM, XMLNSC, or IDOC as the domain.
   - If you are using the MRM parser, select the correct message type from the list in Message type. This list is populated with available message types when you select the MRM parser.
   - If you are using the MRM or IDOC parser, select the correct message format from the list in Message format. This list is populated with available message formats when you select the MRM or IDOC parser.
   - Specify the message coded character set ID in Message coded character set ID.
   - Select the message encoding from the list in Message encoding or specify a numeric encoding value. For more information about encoding, see “Converting data with message flows” on page 165.

7. On the **Parser Options** sub-tab:
   - Parse timing is, by default, set to On Demand, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see “Parsing on demand” on page 1449.
   - If you are using the XMLNSC parser, set values for the properties that determine how the XMLNSC parser operates. For more information, see “Manipulating messages in the XMLNSC domain” on page 408.

8. Use the **Records and Elements** tab to specify how the data is interpreted as records. Only one record is retrieved each time the TCPIPServerReceive node is invoked; therefore, if the TCP/IP stream contains multiple logical messages, you must invoke the node multiple times to receive all the messages.
   - Use the Record detection property to determine how the data is split into records, each of which generates a single message. Choose from the following options:
     - Connection closed specifies that all of the data sent during a connection is a single record.
     - Fixed Length specifies that each record is a fixed number of bytes in length. Each record contains the number of bytes specified in the Length property, except possibly a shorter final record in the file.
     - Select Delimited if the records that you are processing are separated, or terminated, by a DOS or UNIX line end or by a sequence of user-defined delimiter bytes. Specify the delimiter and delimiter type in the Delimiter and Delimiter type properties.
     - Select Parsed Record Sequence if the data contains a sequence of one or more records that are serially recognized by the parser specified in Message domain. The node propagates each recognized record as a separate message. If you select this Record detection option, the parser specified in Message domain must be either XMLNSC or MRM (either CWF or TDS physical format).
   - If you set Record detection to Fixed Length, use Length to specify the required length of the output record. This value must be between 1 byte and 100 MB. The default is 80 bytes.

     If you set Record detection to Connection closed, Fixed Length, or Delimited, a limit of 100 MB applies to the length of the records. If you set Record detection to Parsed Record Sequence, the TCPIPServerReceive node does not determine or limit the length of a record. Nodes that are downstream in the message flow might try to determine the record length or process a long record. If you intend to process large records in this way, ensure that your broker has sufficient memory. You might need to apply message flow techniques described in the Large Messaging sample to make the best use of the available memory.
   - If you set Record detection to Delimited, use Delimiter to specify the delimiter to be used. Choose from the following options:
     - DOS or UNIX Line End, on UNIX systems, specifies the line feed character (<LF>, X'0A'), and, on Windows systems, specifies a carriage return character followed by a line feed character (<CR><LF>, X'0D0A'). The node treats both of these strings as delimiters, irrespective of the system on which the broker is running. If both strings can be seen in the same record, the node recognizes both as delimiters. The node does not recognize X'15' which, on z/OS systems, is the 'newline' byte; set this property to Custom Delimiter and set Custom delimiter to 15 if your input file is coded using EBCDIC new lines.
     - Custom Delimiter (hexadecimal), permits a sequence of bytes to be specified in Custom delimiter (hexadecimal)
   - In Custom delimiter (hexadecimal), specify the delimiter byte or bytes to be used when Delimiter is set to Custom delimiter (hexadecimal). Specify this value as an even-numbered string of hexadecimal digits. The default is X'0A' and the maximum length of the string is 16 bytes (represented by 32 hexadecimal digits).

- If you set Record detection to Delimited, use Delimiter type to specify the type of delimiter. Permitted values are:
  - Infix. If you select this value, each delimiter separates records. If the data ends with a delimiter, the (zero length) data following the final delimiter is still propagated, although it contains no data.
  - Postfix. If you specify this value, each delimiter terminates records. If the data ends with a delimiter, no empty record is propagated after the delimiter. If the data does not end with a delimiter, it is processed as if a delimiter follows the final bytes of the data. Postfix is the default value.
- The TCPIPServerReceive node considers each occurrence of the delimiter in the input as either separating (infix) or terminating (postfix) each record. If the data begins with a delimiter, the node treats the (zero length) contents preceding that delimiter as a record and propagates an empty record to the flow. The delimiter is never included in the propagated message.

9. Use the **Validation** tab to provide validation based on the message set for predefined messages. For more information about validation, see "Validating messages" on page 204. For information about how to complete this tab, see "Validation tab properties" on page 1446.

Terminals and properties

The terminals of the TCPIPServerReceive node are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Out	The output terminal to which the message is routed if it is successfully retrieved from an external resource. If no errors occur within the input node, a message received from an external resource is always sent to the Out terminal first.
Timeout	The terminal to which a message is sent when the time specified in the Timeout waiting for a data record property has been exceeded. The message text is `Timeout value is exceeded`.
Failure	The output terminal to which the message is routed if an error occurs. These errors include failures caused by retry processing. Even if the Validation property is set, messages propagated to this terminal are not validated.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the TCPIPServerReceive node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	TCPIPServerReceive	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the TCPIPServerReceive node determine how the TCP/IP connection is controlled, and are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Connection details	Yes	Yes		A string containing the port number to be used, or the name of a configurable service. The following formats are supported: • Configurable service name. This value is used to look up the port and host name in configurable services. For example, `TCPIPProfile1`. • <Port>. This value is the port number; for example, 1111. • <Port>. This value is the port number. In this case, the host name is assumed to be localhost.	connectionDetails
Timeout waiting for a data record (seconds)	Yes	Yes	60	Specifies how long the node listens on a connection for more data after the first byte of data has arrived. You can specify any length of time in seconds. The default is 60 seconds. When the specified time has been exceeded, all available data is sent to the Failure terminal.	timeoutWaitingForData

The Advanced properties of the TCPIPServerReceive node determine how the data stream is controlled, and are described in the following table.

Property	M	C	Default	Description
Close connection	Yes	No	No	Controls when the connection is closed, or if it remains open. Valid options are: • Select No to leave the connection open. This value is the default. • Select After timeout to close the connection when a timeout occurs. • Select After data has been received to close the connection when the end of the record is found.
Close input stream after a record has been received	Yes	No	Cleared	Specifies whether to close the input stream as soon as the data has been retrieved. When the connection input stream is reserved, no other node can use it without knowing the ID. By default this property is not selected.

Property	M	C	Default	Description
Input Stream Modification	No	No	Leave unchanged	Specifies whether to reserve the input stream for use only by input and receive nodes that specify the connection ID, or to release the input stream at the end of the flow. Valid options are: • Select Leave unchanged to leave the input stream as it was when it entered the node. This value is selected by default. • Select Release input stream to specify that this input stream is returned to the pool and is available for use by any input or receive node. • Select Reserve input stream (for use by future TCPIP input and receive nodes) to specify that this input stream can be used only by this node and by other input or receive nodes that request it by specifying the connection ID. When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID. • Select Reserve input stream (for use by future TCPIP input and receive nodes) then release after propagate to specify that this input stream can be used only by this node and receive nodes that request it by specifying the correct connection ID. After the message has been propagated, this input stream is returned to the pool and becomes available for use by any input or receive node.
Output Stream Modification	No	No	Leave unchanged	Specifies whether to reserve this output stream or release it and return it to the pool for use by any output node. These options are available only if you have not selected the Close output stream after a record has been sent property. • Select Leave unchanged to leave the output stream as it was when it entered the node. This value is selected by default. • Select Release output stream to specify that this output stream is returned to the pool and is available for use by any output node. • Select Reserve output stream (for use by future TCPIP output nodes) to specify that this output stream can be used only by this node and by other output nodes that request it by specifying the connection ID. When the connection input stream is reserved, no other nodes can use it without specifying the correct connection ID. • Select Reserve output stream (for use by future TCPIP output nodes) then release after propagate to specify that this output stream can be used only by this node and output nodes that request it by specifying the correct connection ID. After the message has been propagated, this output stream is returned to the pool and becomes available for use by any output node.

The Request properties of the TCPIPServerReceive node specify the location of the data to be written. You can specify the properties on this tab as XPath or ESQL expressions. Content Assist is available in the Properties view and also in the XPath Expression Builder, which you can run by clicking **Edit** to the right of each property. The Request properties are described in the following table:

Property	M	C	Default	Description
Port location	Yes	No	$LocalEnvironment/TCPIP/Receive/Port	The message element location that contains the Port. Specify the location of the value to override the Port that is set in the Connection details property of the **Basic** tab. If you do not specify a location, the default value is $LocalEnvironment/TCPIP/Receive/Port.
ID location	Yes	No	$LocalEnvironment/TCPIP/Receive/Id	The message element location that contains the ID. Specify the location of the Id of the socket that is being used. This internal identifier is used by WebSphere Message Broker to uniquely identify a connection. If you do not specify a location, the default value is $LocalEnvironment/TCPIP/Receive/Id.
Reply ID location	Yes	No	$LocalEnvironment/TCPIP/Receive/ReplyId	The message element location that contains the Reply ID. Specify the location of the Reply ID that is stored on the connection being used. The Reply ID can be used when data is returned in an input node. If you do not specify a location, the default value is $LocalEnvironment/TCPIP/Receive/ReplyId.

The Result properties of the TCPIPServerReceive node determine where the reply is to be stored, and are described in the following table:

Property	M	C	Default	Description
Output data location	No	No	$OutputRoot	The start location in the output message tree where the parsed elements from the bit string of the message are stored.
Copy local environment	No	No	Selected	Specifies whether the local environment is copied to the output message. • If Copy local environment is selected, a new copy of the local environment is created in the tree, and it is populated with the contents of the local environment from the preceding node. Therefore, that if a node changes the local environment, the upstream nodes are not affected by those changes because they have their own copies. This value is the default. • If Copy local environment is not selected, the node does not generate its own copy of the local environment, but uses the local environment that is passed to it by the preceding node. Therefore, if a node changes the local environment, the changes are reflected by the upstream nodes.

The Input Message Parsing properties of the TCPIPServerReceive node determine how to parse the incoming message.

If the incoming message has an MQRFH2 header, you do not need to set values for the Input Message Parsing properties because the values are derived from the <mcd> folder in the MQRFH2 header; for example:

```
<mcd><Msd>MRM</Msd><Set>DHM4UO906S001</Set><Type>receiptmsg1</Type>
<Fmt>XML</Fmt></mcd>
```

If you set values, and if they differ from the values in the MQRFH2 header, the values in the MQRFH2 header take precedence. The Input Message Parsing properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Message domain	No	No	BLOB	The domain that is used to parse the incoming message. The default is BLOB. You can choose from the following options: • MRM • XMLNSC • DataObject • XMLNS • JMSMap • JMSStream • MIME • BLOB • XML (this domain is deprecated; use XMLNSC) • IDOC (this domain is deprecated; use MRM) You can also specify a user-defined parser, if appropriate.	
Message set	No	No		If you are using the MRM or IDOC parser, or the XMLNSC parser in validating mode, select the Message set in which the incoming message is defined. The list contains the message sets that are available when you select MRM, XMLNSC, or IDOC as the domain. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the Message Set property, or restore the reference to this message set project.	
Message type	No	No		The name of the incoming message. If you are using the MRM parser, select the correct message type from the list in Message type. This list is populated with available message types when you select the MRM parser.	
Message format	No	No		The name of the physical format of the incoming message. If you are using the MRM or IDOC parser, select the correct message format from the list in Message format. This list is populated with available message formats when you select the MRM or IDOC parser.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Message coded character set ID	Yes	No	Broker System Default	The ID of the coded character set used to interpret the data being read.	messageCodedCharSetIdProperty
Message encoding	Yes	No	Broker System Determined	The encoding scheme for numbers and large characters used to interpret the data being read. Valid values are Broker System Determined or a numeric encoding value. For more information about encoding, see "Converting data with message flows" on page 165.	messageEncodingProperty

The Parser Options properties of the TCPIPServerReceive node are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	On Demand	This property controls when an input message is parsed. Valid values are: • On Demand • Immediate • Complete Parse timing is, by default, set to On Demand, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see "Parsing on demand" on page 1449.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the syntax elements in the message tree have data types taken from the XML Schema.
Use XMLNSC compact parser for XMLNS domain	No	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the message data appears under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or Input Message Parsing property, Message Domain, is XMLNS.
Retain mixed content	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in an input message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain comments	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in an input message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain processing instructions	No	No	Cleared	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in an input message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the input message that are to be opaquely parsed by the XMLNSC parser.

The Records and Elements properties of the TCPIPServerReceive node specify how the data is interpreted as records, and are described in the following table:

Property	M	C	Default	Description
Record detection	Yes	No	Connection closed	The mechanism used to identify records in the input data. • Connection closed specifies that all of the data sent during a connection is a single record. • Fixed Length specifies that each record is a fixed number of bytes in length. Each record contains the number of bytes specified in the Length property, except possibly a shorter final record in the file. • Select Delimited if the records you are processing are separated, or ended, by a DOS or UNIX line end or by a sequence of user-defined delimiter bytes. Specify the delimiter and delimiter type in the Delimiter and Delimiter type properties. • Select Parsed Record Sequence if the data contains a sequence of one or more records that are serially recognized by the parser specified in Message domain. The node propagates each recognized record as a separate message. If you select this Record detection option, the parser specified in Message domain must be either XMLNSC or MRM (either CWF or TDS physical format).
Length (bytes)	Yes	No	0	If you set Record detection to Fixed Length, use Length to specify the required length of the output record in bytes. This value must be between 1 byte and 100 MB. The default is 80 bytes. If you set Record detection to Connection closed, Fixed Length, or Delimited, a limit of 100 MB applies to the length of the records. If you set Record detection to Parsed Record Sequence, the TCPIPServerReceive node does not determine or limit the length of a record. Nodes that are downstream in the message flow might try to determine the record length or process a long record. If you intend to process large records in this way, ensure that your broker has sufficient memory. You might need to apply message flow techniques described in the Large Messaging sample to make the best use of the available memory.
Delimiter	Yes	No	DOS or UNIX Line End	If you set Record detection to Delimited, use Delimiter to specify the delimiter to be used. Choose from the following options: • DOS or UNIX Line End, on UNIX systems, specifies the line feed character (<LF>, X'0A'), and, on Windows systems, specifies a carriage return character followed by a line feed character (<CR><LF>, X'0D0A'). The node treats both of these strings as delimiters, irrespective of the system on which the broker is running. If both strings are displayed in the same record, the node recognizes both as delimiters. The node does not recognize X'15' which, on z/OS systems, is the 'newline' byte; set this property to Custom Delimiter and set Custom delimiter to 15 if your input file is coded using EBCDIC new lines, such as EBCDIC files from a z/OS system. • Custom Delimiter (hexadecimal), permits a sequence of bytes to be specified in Custom delimiter (hexadecimal)
Custom delimiter (hexadecimal)	No	No		The delimiter byte or bytes to be used when Delimited record detection and Custom Delimiter (Hexadecimal) are selected. This property is mandatory only if the Delimiter property is set to Custom Delimiter (Hexadecimal). Specify this value as an even-numbered string of hexadecimal digits. The default is X'0A' and the maximum length of the string is 16 bytes (represented by 32 hexadecimal digits).

Property	M	C	Default	Description
Delimiter type	Yes	No	Postfix	The location of the delimiter when Delimited record detection and Custom Delimiter (Hexadecimal) are selected. Valid options are: • Infix. If you select this value, each delimiter separates records. If the data ends with a delimiter, the (zero length) data following the final delimiter is still propagated, although it contains no data. • Postfix. If you specify this value, each delimiter ends records. If the data ends with a delimiter, no empty record is propagated after the delimiter. If the data does not end with a delimiter, it is processed as if a delimiter follows the final bytes of the data. Postfix is the default value. The TCPIPServerReceive node considers each occurrence of the delimiter in the input as either separating (infix) or terminating (postfix) each record. If the data begins with a delimiter, the node treats the (zero length) contents preceding that delimiter as a record and propagates an empty record to the flow. The delimiter is never included in the propagated message. This property is ignored unless the Delimiter property is set to Custom Delimiter (Hexadecimal).

The Validation properties of the TCPIPServerReceive node are described in the following table.

For a full description of these properties, see "Validation properties" on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are • None • Content and Value • Content • Inherit	validateMaster
Failure action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are: • User Trace • Local Error Log • Exception • Exception List	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Throw node

Use the Throw node to throw an exception in a message flow.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow"
- "Terminals and properties"

Purpose

An exception can be caught and processed by:

- A preceding TryCatch node
- The message flow input node (the built-in nodes, for example HTTPInput and MQInput, have Catch terminals)
- A preceding AggregateReply node

Include a Throw node to force an error path through the message flow if the content of the message contains unexpected data. For example, to back out a message that does not contain a particular field, you can check (using a Filter node) that the field exists; if the field does not exist, the message can be passed to a Throw node that records details about the exception in the exception list subtree in the message.

The Throw node is contained in the **Construction** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Look at the following samples to see how to use this node:
- Airline Reservations
- Error Handler
- Large Messaging

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

Include a Throw node with a TryCatch node in your message flow to alert the systems administrator of a potential error situation; for example, if you have a Compute node that calculates a number, test the result of this calculation and throw an exception if the result exceeds a certain amount. The TryCatch node catches this exception and propagates the message to a sequence of nodes that process the error.

Terminals and properties

When you have put an instance of the Throw node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The Throw node terminal is described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Throw node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type: Throw	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Throw node Basic properties are described in the following table.

Property	M	C	Default	Description
Message Catalog	No	No		The name of the message catalog from which the error text for the error number of the exception is extracted. Enter the fully-qualified path and file name of the message catalog that contains the message source. This file can be your own message catalog, or the default message catalog that is supplied with WebSphere Message Broker. To use the default supplied catalog, leave this property blank.
Message Number	No	No	3001	The error number of the exception that is being thrown. • If you have created your own message catalog, enter the number for the message in the catalog that you want to use when this exception is thrown. • If you are using the default message catalog, specify a number between 3001 (the default) and 3049. These numbers are reserved in the default catalog for your use. The text of each of these messages in the default message catalog is identical, but you can use a different number in this range for each situation in which you throw an exception; use the number to identify the exact cause of the error.
Message Text	No	No		Additional text that explains the cause of the error. Enter any additional free format text that contains information that you want to include with the message when it is written to the local error log; for example, if you have checked for the existence of a particular field in a message and thrown an exception when that field is not found, you might include the text: `The message did not contain the required field: Branch number` If you are using the default message catalog, this text is inserted as `&1` in the message text.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

TimeoutControl node

Use the TimeoutControl node to process an input message that contains a timeout request.

This topic contains the following sections:

Purpose

The TimeoutControl node validates the timeout request message, stores the message, and propagates the message (unchanged) to the next node in the message flow. For more information, see "Sending timeout request messages" on page 696.

The TimeoutControl node is contained in the **Timer** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Use a TimeoutControl node and a TimeoutNotification node together in a message flow for an application that requires events to occur at particular times, or at regular intervals.

Examples of when you can use the timeout nodes in a message flow include:
- You need to run a batch job every day at midnight.
- You want information about currency exchange rates to be sent to banks at hourly intervals.
- You want to confirm that important transactions are processed within a certain time period and perform some other specified actions to warn when a transaction has not been processed in that time period.

You can use more than one TimeoutControl node with a TimeoutNotification node. Timeout requests that are initiated by those TimeoutControl nodes are all processed by the same TimeoutNotification node if the same Unique identifier is used for the TimeoutNotification node and each of the TimeoutControl nodes.

Look at the following sample for more details about how to use the timeout processing nodes:
- Timeout Processing

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

Terminals and properties

When you have put an instance of the TimeoutControl node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The TimeoutControl node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message tree for processing (which includes validating the timeout request specified in the message tree at Request location) and adds it to the control queue.
Failure	The output terminal to which the input message is propagated if a failure is detected during processing in this node. If this terminal is not connected to another node, error information is passed back to the previous node in the message flow.
Out	The output terminal to which incoming messages are propagated, unchanged, after successful timeout request processing. If this terminal is not connected to another node, no propagation occurs. If propagation of the message fails, the message is propagated to the Failure terminal.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the TimeoutControl node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, TimeoutControl	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the TimeoutControl node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Unique identifier	Yes	Yes	None	This is the only mandatory property for the node. Its value must be unique within the broker. The equivalent property of the TimeoutNotification node with which it is paired must have the same value. The maximum length of this identifier is 12 characters. This name is also used to identify a Timer configurable service (if one exists) to be used by the node.	uniqueIdentifier
Request location	No	No	None	This property describes where to find the timeout request information in the incoming message. This value can be any valid location in the input message tree and is validated at run time. If you do not specify a request location, InputLocalEnvironment.TimeoutRequest is assumed. For more information about the timeout request message, see "Sending timeout request messages" on page 696.	
Request persistence	No	No	Automatic	This property controls whether an incoming timeout request survives a restart of either the broker or the message flow that contains the TimeoutNotification node that is paired with this TimeoutControl node. Select Yes if you want the incoming request to persist; select No if you do not. If you select Automatic (the default), the Persistence setting in the Properties folder of the incoming message is used.	

The Message properties of the TimeoutControl node are described in the following table.

Property	M	C	Default	Description
Stored message location	No	No	None	This property identifies the location of the part of the request message that you want to store for propagation by the TimeoutNotification node with which this node is paired. If you do not specify a value, the entire message is stored. You can specify any valid location in the message tree. If you choose to store the entire message, you do not need to specify any values in Message domain, Message set, Message type, or Message format.

Property	M	C	Default	Description
Message domain	No	No	BLOB	The domain that is used to parse the stored timeout request message by the TimeoutNotification node. If you do not specify a value and the message location is stored, the default value is BLOB. Select the name of the parser that you are using. This value, and the three corresponding values in Message set, Message type, and Message format, are used by the TimeoutNotification node with which it is paired when it rebuilds the stored message for propagation. If you have stored the entire request message (by leaving Stored message location blank), do not specify any values here. If you choose to store part of the request message, specify values here that reflect the stored request message fragment as if it were the entire message, which is the case when it is processed by the TimeoutNotification node. Choose from the following parsers: • MRM • XMLNSC • XMLNS • BLOB • XML (this domain is deprecated; use XMLNSC) You can also specify a user-defined parser, if appropriate.
Message set	No	No	None	The name or identifier of the message set in which the stored timeout request message is defined. If you are using the MRM parser, or the XMLNSC parser in validating mode, select the Message set that you want to use from the list. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Message type	No	No	None	The name of the stored timeout request message. If you are using the MRM parser, select the correct message from the list in Message type. This list is populated with messages that are defined in the Message set that you have selected.
Message format	No	No	None	The name of the physical format of the stored timeout request message. If you are using the MRM parser, select the format of the message from the list in Message format. This list includes all the physical formats that you have defined for this Message set.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

TimeoutNotification node

Use the TimeoutNotification node to manage timeout-dependent message flows.

This topic contains the following sections:

Purpose

The TimeoutNotification node is an input node that you can use in two ways:

- A TimeoutNotification node can be paired with one or more TimeoutControl nodes.

  The TimeoutNotification node processes timeout request messages that are sent by the TimeoutControl nodes with which it is paired, and propagates copies of the messages (or selected fragments of the messages) to the next node in the message flow.
- A TimeoutNotification node can be used as a stand-alone node.

  Generated messages are propagated to the next node in the message flow at time intervals that are specified in the configuration of this node.

The TimeoutNotification node is contained in the **Timer** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Use a TimeoutControl node and a TimeoutNotification node together in a message flow for an application that requires events to occur at a particular time, or at regular intervals; for example, when you want a batch job to run every day at midnight, or you want information about currency exchange rates to be sent to banks at hourly intervals.

You can use more than one TimeoutControl node with a TimeoutNotification node. Timeout requests that are initiated by those TimeoutControl nodes are all processed by the same TimeoutNotification node if the same Unique Identifier is used for the TimeoutNotification node and each of the TimeoutControl nodes. However, do not use the same Unique Identifier for more than one TimeoutNotification node.

Timeout request messages are stored for processing on a queue used by the TimeoutNotification node. By default, this queue is the SYSTEM.BROKER.TIMEOUT.QUEUE. However, you can use a Timer configurable service to specify an alternative timeout queue, which provides greater control over the storage of messages. For information about using an alternative timeout queue, see “Configuring the storage of events for timeout nodes” on page 655.

When a TimeoutNotification node is started as a result of the broker starting, or by the message flow that contains the node starting, it scans its internal timeout store and purges any non-persistent timeout requests. Notifications are issued for any persistent timeout requests that are now past and that have the IgnoreMissed property set to `False`.

If you use a TimeoutNotification node to generate a WebSphere MQ message to an output node, such as theMQOutput node, provide a valid MQMD. You must also provide a valid MQMD if the TimeoutNotification node is running in automatic mode (as a stand-alone node). If the TimeoutNotification node is running in controlled mode (that is, it is paired with one or more TimeoutControl nodes), you must provide a valid MQMD only if the stored messages do not already have an MQMD. The following ESQL shows how to provide a valid MQMD:

```
CREATE NEXTSIBLING OF OutputRoot.Properties DOMAIN 'MQMD';
SET OutputRoot.MQMD.Version = MQMD_CURRENT_VERSION;
SET OutputRoot.MQMD.Format = 'XML';
```

Look at the following sample for more details about how to use the timeout processing nodes:

- Timeout Processing

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

Terminals and properties

When you have put an instance of the TimeoutNotification node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The terminals of the TimeoutNotification node are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is propagated if a failure is detected during processing in this node. Nodes can be connected to this terminal to process these failures. If this terminal is not connected to another node, messages are not propagated and no logging or safe storage of data occurs.
Out	The output terminal to which messages are propagated after timeouts expire. • If the TimeoutNotification node is running in Automatic mode (that is, there are no TimeoutControl nodes paired with this node), the propagated messages contain only a Properties folder and a LocalEnvironment that is populated with the timeout information. • If the TimeoutNotification node is running in Controlled mode (that is, TimeoutControl nodes that are paired with this node store timeout requests), the propagated messages contain what was stored by the TimeoutControl nodes, which might be entire request messages or fragments of them. If the TimeoutNotification node is used as the input node to a message flow that generates a WebSphere MQ message (for example, by using an MQOutput node), the message flow must create the necessary MQ headers and data (for example, MQMD).
Catch	The output terminal to which the message is propagated if an exception is thrown downstream. If this terminal is not connected to another node, the following events occur: 1. The TimeoutNotification node writes the error to the local error log. 2. The TimeoutNotification node repeatedly tries to process the request until the problem that caused the exception is resolved.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the TimeoutNotification node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type: TimeoutNotification	The name of the node.
Short description	No	No		A brief description of the node.

Property	M	C	Default	Description
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the TimeoutNotification node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Unique Identifier	Yes	Yes	None	This property specifies a value that is unique within the broker and that is the same as the identifier that is specified for the TimeoutControl nodes with which this node is paired (if there are any). The maximum length of this identifier is 12 characters. This name is also used to identify a Timer configurable service (if one exists) to be used by the node. Do not use the same Unique Identifier for more than one TimeoutNotification node.	uniqueIdentifier
Transaction Mode	No	No	Yes	The transaction mode for the node. If the transaction mode is Automatic, a transaction is based on the persistence of the stored messages, which is controlled by the Request Persistence property of the TimeoutControl node with which it is paired. You can set this property to one of the following values: • Select Yes if you want a transaction to be started. • Select No if you do not want a transaction to be started. • Select Automatic only if you have set Operation Mode to Controlled. Whether a transaction is started depends on the persistence of the stored timeout requests, which is controlled by the value of Request Persistence in the TimeoutControl node with which it is paired.	
Operation Mode	No	No	Automatic	This property indicates whether this node is paired with any paired TimeoutControl nodes. Valid values are: • If you select Automatic the node is not paired with any TimeoutControl nodes. The node generates timeout requests with an interval that is controlled by the setting of the Timeout Value property, which must be a positive integer. • If you select Controlled the node processes all timeout requests that have been stored by the TimeoutControl nodes with which it is paired.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Timeout Interval	No	No	1	The interval (in seconds) between timeout requests. This property is relevant only if Operation Mode is set to Automatic. The value of this property must be a positive integer. If the Operation Mode is set to Automatic, the value of the Timeout Interval property is overridden by the **Timeout interval** property, if set, in the Timer configurable service.	

The properties of the Parser Options for the TimeoutNotification node are described in the following table.

Property	M	C	Default	Description
Parse Timing	No	No	On Demand	This property controls when the timeout message is parsed. Valid values are On Demand, Immediate, and Complete. By default, this property is set to On Demand, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see "Parsing on demand" on page 1449.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the XMLNSC parser creates syntax elements in the message tree with data types taken from the XML Schema. You can select this property only if you set the Validate property on the **Validation** tab to Content or Content and Value.
Use MQRFH2C Compact Parser for MQRFH2 Domain	No	No	Cleared	This property controls whether the MQRFH2C Compact Parser, instead of the MQRFH2 parser, is used for MQRFH2 headers.
Use XMLNSC Compact Parser for XMLNS Domain	No	No	Cleared	This property controls whether the XMLNSC Compact Parser is used for messages in the XMLNS Domain. If you set this property, the message data appears under XMLNSC in nodes that are connected to the output terminal when the input RFH2 header or default properties Domain is XMLNS.
Retain Mixed Content	No	No	None	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters mixed text in a timeout message. If you select the check box, elements are created for mixed text. If you clear the check box, mixed text is ignored and no elements are created.
Retain Comments	No	No	None	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters comments in a timeout message. If you select the check box, elements are created for comments. If you clear the check box, comments are ignored and no elements are created.
Retain Processing Instructions	No	No	None	This property controls whether the XMLNSC parser creates elements in the message tree when it encounters processing instructions in a timeout message. If you select the check box, elements are created for processing instructions. If you clear the check box, processing instructions are ignored and no elements are created.
Opaque elements	No	No	Blank	This property is used to specify a list of elements in the timeout message that are to be opaquely parsed by the XMLNSC parser. Opaque parsing is performed only if validation is not enabled (that is, if Validate is None); entries that are specified in Opaque Elements are ignored if validation is enabled.

The Validation properties of the TimeoutNotification node are described in the following table.

If a message is propagated to the Failure terminal of the node, it is not validated. For more information, see "Validating messages" on page 204 and "Validation properties" on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are None, Content, and Content And Value.	validateMaster
Failure Action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Trace node

Use the Trace node to generate trace records that you can use to monitor the behavior of a message flow.

This topic contains the following sections:

Purpose

Trace records can incorporate text, message content, and date and time information, to help you to monitor the behavior of the message flow.

You can write the records to the user trace file, another file, or the local error log (which contains error and information messages written by all other WebSphere Message Broker components). If you write traces to the local error log, you can issue a message from the default message catalog that is supplied with WebSphere Message Broker, or you can create your own message catalog.

The operation of the Trace node is independent of the setting of user tracing for the message flow that contains it. In particular, records that are written by the Trace node to the user trace log are written even if user trace is not currently active for the message flow.

The Trace node is contained in the **Construction** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Look at the following samples to see how to use this node:
- Airline Reservations
- Aggregation
- Timeout Processing

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

Include a Trace node to help diagnose errors in your message flow. By tracing the contents of the message at various points in the flow, you can determine the sequence of processing. You can configure the Trace node to record the content of a message, and to check the action of a specific node on the message. For example, you can include a Trace node immediately after a Compute node to check that the output message has the expected format.

You can also use the Trace node to provide information in error handling in your message flows. For example, you can use this node to record failures in processing because of errors in the content or format of a message.

When you have tested the message flow and proved that its operation is correct, remove Trace nodes from your message flow, or switch them off.

Terminals and properties

When you have put an instance of the Trace node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The terminals of the Trace node are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Out	The output terminal through which the message is propagated.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the Trace node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type: Trace	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the Trace node are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Destination	Yes	No	User Trace	The destination of the trace record that is written by the node. The Destination refers to the computer that hosts the broker on which the Trace node is deployed: • To write the trace record to the local system error log, select Local Error Log. The information that you include in the trace record is written to one of the following locations: – Windows On Windows systems, data is written to the Event log (Application View) – Linux UNIX On Linux and UNIX systems, data is written to the `syslog` – z/OS On z/OS systems, data is written to the operator console UNIX On UNIX systems, syslog entries are restricted in length and messages are truncated by the newline character. To record a large amount of data in a log, set the destination to File or User Trace instead. If you select Local Error Log, indicate the number of the trace message that is to be written, and the message catalog in which the message is defined. – If you leave Message Catalog blank, the default message catalog is used as the source of the message that is to be written. You must also enter the error number of the record in Message Number. Numbers 3051 to 3099 are reserved in the default catalog for this use. The text of each of these messages in the default message catalog is identical, but if you use a different number in this range for each situation that you trace, you can identify the exact cause of the error. The default message number is 3051. – If you create your own message catalog, enter the fully qualified file name for your catalog in Message Catalog. You must also enter the appropriate number for the message in the catalog that you want to write to the local error log in Message Number. On some systems, message numbers that end `00` are reserved for system use; do not include messages with numbers such as 3100 in your message catalog.	

Property	M	C	Default	Description	mqsiapplybaroverride command property

				• To write the trace record to the system-generated user trace log, select User Trace.
				These records are written regardless of the setting of the User Trace property for the deployed message flow.
				The location of the trace logs depends on your environment:
				Windows **Windows**
				If you set the work path using the **-w** parameter of the mqsicreatebroker command, the location is `workpath\log`.
				If you have not specified the broker work path, the location is:`%ALLUSERSPROFILE%\Application Data\IBM\MQSIcommon\log` where `%ALLUSERSPROFILE%` is the environment variable that defines the system working directory. The default directory depends on the operating system:
				– On Windows XP and Windows Server 2003: `C:\Documents and Settings\All Users\Application Data\IBM\MQSIcommon\log`
				– On Windows Vista and Windows Server 2008: `C:\ProgramData\IBM\MQSIcommon\log`
				The actual value might be different on your computer.
				Linux UNIX **Linux and UNIX**
				`/var/mqsi/common/log`
				z/OS **z/OS**
				`/component_filesystem/log`
				The file name is made up of the broker name, the broker UUID, and a suffix of `userTrace.bin` (for example, `broker.e51906cb-dd00-0000-0080-b10e69a5d551.userTrace.bin.0`). Use the mqsireadlog and mqsiformatlog commands before you view the user trace log.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
				• To write the trace record to a file of your choice, select File. If you select this option, you must also set File Path to the fully qualified path name for the trace. If you do not set the path, the location of the file depends on the system; for example, on z/OS, the file is created in the home directory of the broker service ID. You can use any name for the trace file; for example, `c:\user\trace\trace.log` If you specify a file that does not exist already, the file is created. However, directories are not created by this process, so the full path must exist. The file is written as text, in the format specified by the Pattern property. You do not need to run the mqsireadlog or mqsiformatlog commands against the file. If a file write error occurs during processing (because of an out of space condition, for example), a single warning message, BIP4065, is written to the local system error log, and the message flow continues to process messages without logging further errors. Check the error log carefully for such a message. • If you do not want to write trace records, select None. You can also switch off Trace nodes.	
File Path	No	Yes		The fully qualified file name of the file to which to write records. This property is valid only if Destination is set to File.	filePath

Property	M	C	Default	Description	mqsiapplybaroverride command property

Pattern	No	No		The data that is to be included in the trace record. Create an ESQL pattern to specify what information to write. If you write the trace record to the local error log, the pattern governs the information that is written in the text of the message number that is selected. If you use the default message catalog, and a number between 3051 and 3099, the pattern information is inserted as &1 in the message text.
				• You can write plain text, which is copied into the trace record exactly as you have entered it.
				• You can identify parts of the message to write to the trace record, specifying the full field identifiers enclosed between the characters ${ and }. To record the entire message, specify ${Root}.
				• Use the ESQL functions to provide additional information; for example, use the ESQL function CURRENT_DATE to record the date, time, or both, at which the trace record is written.
				
				The pattern shown here includes some of the options that are available. The pattern writes an initial line of text, records two elements of the current message, and adds a simple timestamp:
				`Message passed through with the following fields:`
				`Store name is ${Body.storedetailselement.storename}`
				`Total sales are ${Body.totalselement.totalsales}`
				`Time is: ${EXTRACT(HOUR FROM CURRENT_TIMESTAMP)}`
				`:${EXTRACT(MINUTE FROM CURRENT_TIMESTAMP)}`
				
				The resulting trace record is:
				`Message passed through with the following fields:`
				`Store name is 'SRUCorporation'`
				`Total sales are '34.98'`
				`Time is: 11:19`
				
				A pattern that contains syntax errors does not prevent a message flow that contains the Trace node from deploying, but the node writes no trace records.	

Message Catalog	No	No		The name of the message catalog from which the error text for the error number of the exception is extracted. The default value (blank) indicates that the message is taken from the message catalog that is supplied with WebSphere Message Broker. See Creating message catalogs for more information.	
Message Number	No	No	3051	The error number of the message that is written.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

TryCatch node

Use the TryCatch node to provide a special handler for exception processing.

This topic contains the following sections:

Purpose

Initially, the input message is routed on the Try terminal, which you must connect to the remaining non-error processing nodes of the message flow. If a downstream node (which can be a Throw node) throws an exception, the TryCatch node catches it and routes the original message to its Catch terminal. Connect the Catch terminal to further nodes to provide error processing for the message after an exception. If the Catch terminal is connected, the message is propagated to it. If the Catch terminal is not connected, the message is discarded.

The TryCatch node is contained in the **Construction** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Look at the following sample to see how to use this node:

- Error Handler

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

Use the Throw and TryCatch nodes when you use the Compute node to calculate a total. You can create a message that is sent to your system administrator when the total that is calculated exceeds the maximum value for the Total field.

Connecting the terminals

The TryCatch node has no configurable properties that affect its operation. You determine how it operates by connecting the output terminals to subsequent nodes in your message flow.

1. Connect the Try terminal to the first node in the sequence of nodes that provides the normal (non-error) phase of processing of this message. This

sequence can contain one or more nodes that perform any valid processing. The sequence of nodes can optionally conclude with an output node.

2. Connect the Catch terminal to the first node in the sequence of nodes that provides the error processing for this message flow. This sequence can contain one or more nodes that perform any valid processing. The sequence of nodes can optionally conclude with an output node.

   When an exception is thrown in the message flow, either by the explicit use of the Throw node or the ESQL THROW statement, or by the broker raising an implicit exception when it detects an error that the message flow is not programmed to handle, control returns to the TryCatch node.

   The message is propagated through the Catch terminal and the error handling that you have designed is executed. The message that is propagated through this terminal has the content that it had at the point at which the exception was thrown, including the full description of the exception in the ExceptionList.

Terminals and properties

When you have put an instance of the TryCatch node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view.

The TryCatch node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Catch	The output terminal to which the message is propagated if an exception is thrown downstream and caught by this node.
Try	The output terminal to which the message is propagated if it is not caught.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The TryCatch node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type: TryCatch	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

TwineballInput node

Use the TwineballInput node to discover how the WebSphere Adapters nodes work.

This topic contains the following sections:
- "Purpose"
- "Terminals and properties"

Purpose

The TwineballInput node is provided for educational purposes and helps you to see how the WebSphere Adapters nodes work. The TwineballInput node is a sample node with its own sample EIS. You cannot use the TwineBall nodes to connect to the external SAP, Siebel, and PeopleSoft EIS systems. Do not use this node in production.

The TwineballInput node is contained in the **WebSphere Adapters** drawer of the message flow node palette, and is represented in the workbench by the following icon:

You can use the mqsisetdbparms command in the following format to configure an account name with a user name and password for the Twineball adapter.

```
mqsisetdbparms broker name -n adapter name -u user name -p password
```

For example:

```
mqsisetdbparms BRK1 -n eis::TwineballInbound.inadapter -u mqbroker -p ********
```

Look at the following sample to see how to use this node:
- Twineball Example EIS Adapter

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

Terminals and properties

When you have put an instance of the TwineballInput node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. If you double-click a TwineballInput node, you open the Adapter Connection wizard. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The TwineballInput node terminals are described in the following table.

Terminal	Description
Out	Business object events from the adapter are sent to the Out terminal.
Failure	If an error happens in the TwineballInput node, the message is propagated to the Failure terminal. Information about the error, and business object events can also be propagated to the Failure terminal.
Catch	Business object events are sent to the Catch terminal if they cause an uncaught exception in the message flow. If the Catch terminal is not connected, the retry process is activated to handle the business object.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file at deployment).

The TwineballInput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, TwineballInput.	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The TwineballInput node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Adapter component	Yes	Yes		The name of the adapter component that contains configuration properties for the adapter. Either enter a name of an adapter file or click **Browse** to select an adapter file from the list of files that are available in referenced message set projects.	adapterComponent

The TwineballInput node Routing properties are described in the following table.

Property	M	C	Default	Description
Set destination list	No	No	Selected	This property specifies whether to add the method binding name to the route to label destination list. If you select this check box, the method binding name is added so that you can use a RouteToLabel node in the message flow after the TwineballInput node.
Label prefix	No	No		The prefix to add to the method name when routing to label. Add a label prefix to avoid a clash of corresponding label nodes when you include multiple WebSphere Adapters input nodes in the same message flow. By default, there is no label prefix, so that the method name and label name are identical.

The TwineballInput node Input Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	DataObject	The domain that is used to parse the incoming message. By default, the message that is propagated from the TwineballInput node is in the DataObject domain. You cannot specify a different domain.
Message set	Yes	No	Set automatically	The name of the message set in which the incoming message is defined. This field is set automatically from the Adapter component property. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Message type	No	No		The name of the incoming message. The node detects the message type automatically. You cannot set this property.
Message format	No	No		The name of the physical format of the incoming message. You cannot set this property.

The TwineballInput node Transactional properties are described in the following table.

Property	M	C	Default	Description
Transaction mode	No	No	Yes	The transaction mode on this input node determines whether the rest of the nodes in the flow operate under sync point.

The Instances properties of the TwineballInput node are described in the following table. For a full description of these properties, see "Configurable message flow properties" on page 1324.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Additional instances pool	No	Yes	Use Pool Associated with Message Flow	The pool from which additional instances are obtained. • If you select Use Pool Associated with Message Flow, additional instances are obtained from the message flow value. • If you select Use Pool Associated with Node, additional instances are allocated from the node's additional instances based on the number specified in the Additional instances property.	componentLevel
Additional instances	No	Yes	0	The number of additional instances that the node can start if the Additional instances pool property is set to Use Pool Associated with Node. By default, no additional instances are given to the node.	additionalInstances

The TwineballInput node Retry properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Retry mechanism	No	No	Failure	This property specifies how retry processing is handled when a failure is rolled back to the TwineballInput node. • If you select Failure, retry processing is not performed so that you cannot set the remaining properties. • If you select Short and long retry, retry processing is performed first at the interval that is specified by the Short retry interval property, and if that is unsuccessful, it is then performed at the interval that is specified by the Long retry interval property.	
Retry threshold	No	Yes	0	The maximum number of times that retry processing is performed for short retry.	retryThreshold
Short retry interval	No	Yes	0	The interval between short retry attempts.	shortRetryThreshold
Long retry interval	No	Yes	0	The interval between long retry attempts.	longRetryThreshold

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

TwineballRequest node

Use the TwineballRequest node to discover out how WebSphere Adapters nodes work.

This topic contains the following sections:

Purpose

The TwineballRequest node is provided for educational purposes and helps you to see how the WebSphere Adapters nodes work. The TwineballRequest node is a sample node with its own sample EIS. You cannot use the TwineBall nodes to connect to the external SAP, Siebel, and PeopleSoft EIS systems. Do not use this node in production.

The TwineballRequest node is contained in the **WebSphere Adapters** drawer of the message flow node palette, and is represented in the workbench by the following icon:

You can use the mqsisetdbparms command in the following format to configure an account name with a user name and password for the Twineball adapter.

```
mqsisetdbparms broker name -n adapter name -u user name -p password
```

For example:

```
mqsisetdbparms BRK1 -n eis::TwineballOutbound.outadapter -u mqbroker -p ********
```

Look at the following sample to see how to use this node:

- Twineball Example EIS Adapter

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

Terminals and properties

When you have put an instance of the TwineballRequest node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. If you double-click a TwineballRequest node, you open the Adapter Connection wizard. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The TwineballRequest node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts the request business object.
Out	The output terminal to which the response business object is sent if it represents successful completion of the request, and if further processing is required within this message flow.
Failure	If an error happens in the TwineballRequest node, the message is propagated to the Failure terminal. Information about the error, and business object events can also be propagated to the Failure terminal.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk on the panel if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The TwineballRequest node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, TwineballRequest	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The TwineballRequest node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Adapter component	Yes	No		The name of the adapter component that contains configuration properties for the adapter. Either enter a name of an adapter file, or click **Browse** to select an adapter file from the list of files that are available in referenced message set projects.	
Default method	Yes	Yes		The default method binding to use.	defaultMethod

The TwineballRequest node Response Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	DataObject	The domain that is used to parse the response message. By default, the response message that is propagated from the TwineballRequest node is in the DataObject domain. You cannot specify a different domain.
Message set	Yes	No	Set automatically	The name of the message set in which the incoming message is defined. This field is set automatically from the Adapter component property. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Message type	No	No		The name of the response message. The node detects the message type automatically. You cannot set this property.
Message format	No	No		The name of the physical format of the response message. You cannot set this property.

The TwineballRequest node Transactionality properties are described in the following table.

Property	M	C	Default	Description
Transaction mode	No	No	Automatic	This property specifies how updates are handled. If you select Yes, updates are performed in a single transaction. If you select No, updates are performed independently.

The TwineballRequest node Request properties are described in the following table.

Property	M	C	Default	Description
Method Location	Yes	No	$LocalEnvironment/ Adapter/MethodName	The location of the business method (such as createPurchaseOrder or deletePurchaseOrder) that is used to trigger the TwineballRequest node to perform an action on the external system.
Data Location	Yes	No	$Body	The location in the incoming message tree from which data is retrieved to form the request that is sent from the TwineballRequest node to the EIS.

The TwineballRequest node Result properties are described in the following table.

Property	M	C	Default	Description
Output data location	No	No	$OutputRoot	The message tree location to which the TwineballRequest node sends output.
Copy local environment	No	No	Selected	This property controls how the local environment is copied to the output message. If you select the check box, at each node in the message flow, a new copy of the local environment is created in the tree, and it is populated with the contents of the local environment from the preceding node. If a node changes the local environment, the upstream nodes do not see those changes because they have their own copies. This behavior might be an issue if you are using a FlowOrder node, or if you use the propagate command on a Compute node. If you clear the check box, each node does not generate its own copy of the local environment, but it uses the local environment that is passed to it by the previous node. If a node changes the local environment, those changes are seen by the upstream nodes.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Validate node

Use the Validate node to check that the message that arrives on its input terminal is as expected. You can use this node to check that the message has the expected message template properties (the message domain, message set, and message type) and to check that the content of the message is correct by selecting message validation.

This topic contains the following sections:

Purpose

The checks that you can perform depend on the domain of the message.

Check	Domain
Check message domain	All domains
Check message set	XMLNSC, MRM, and IDOC only
Check message type	MRM only
Validate message body	XMLNSC, MRM and IDOC only

You can check the message against one or more of message domain, message set, or message type. The property is checked only if you select its corresponding check box, which means that a property that contains an empty string can be compared.

You can check the content of the message by giving a value to the Validate property. Validation takes place if the Validate property is set to a value other than None, which is the default value.

For validation failures to be returned to the Validate node from the parser, set the Failure Action property to either Exception or Exception List. Otherwise, validation failures are just logged.

If all the specified checks pass, the message is propagated to the Match terminal of the node.

If any of the checks fail, the message is propagated to the Failure terminal. If the Failure terminal is not connected to some failure handling processing, an exception is generated.

The Validate node replaces the Check node, which is deprecated in WebSphere Message Broker Version 6.0 and subsequent releases. The Validate node works in the same way as the Check node, but it has additional Validation properties to allow the validation of message content by parsers that support that capability.

The Validate node is contained in the **Validation** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

Use the Validate node to confirm that a message has the correct message template properties, and has valid content, before propagating the message to the rest of the flow. Subsequent nodes can then rely on the message being correct, without doing their own error checking.

You can also use the Validate node to ensure that the message is routed appropriately through the message flow. For example, configure the node to direct a message that requests stock purchases through a different route from that required for a message that requests stock sales.

Another routing example is the receipt of electronic messages from your staff at your head office. These messages are used for multiple purposes (for example, to request technical support or stationery, or to advise you about new customer leads). These messages can be processed automatically because your staff complete a standard form. If you want these messages to be processed separately from other messages that are received, use the Validate node to ensure that only staff messages that have a specific message type are processed by this message flow.

Terminals and properties

When you have put an instance of the Validate node into a message flow, you can configure it. For more information, see “Configuring a message flow node” on page 276

page 276. The properties of the node are displayed in the Properties view. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The terminals of the Validate node are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if the incoming message does not match the specified properties.
Match	The output terminal to which the message is routed if the incoming message matches the specified properties.

The following tables describe the properties of the node. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Description properties of the Validate node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	Validate	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Validate node Basic properties are described in the following table.

Property	M	C	Default	Description
Domain	No	No		The name of the domain. Select one of the following values from the list of the Domain property: • MRM • SOAP • XMLNSC • DataObject • XMLNS • JMSMap • JMSStream • MIME • BLOB • XML (deprecated - use XMLNSC) • IDOC (deprecated - use MRM) You can also specify a user-defined parser, if appropriate.
Check domain	Yes	No	Cleared	If you select this check box, the incoming message is checked against the Domain property.

Property	M	C	Default	Description
Set	No	No		The name or identifier of the message set to which the incoming message belongs. If you are using the XMLNSC, MRM, or IDOC parser and want to check that the incoming message belongs to a particular message set, select Check set and select one of the values from the list of the Set property. This list is populated when you select XMLNSC, MRM, or IDOC as the message domain. Leave Set clear for the other parsers. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the Set property, or restore the reference to this message set project.
Check set	Yes	No	Cleared	If you select the check box, the incoming message is checked against the Set property. If you are using the XMLNSC, MRM, or IDOC parser and want to check that the incoming message belongs to a particular message set, select Check set and select one of the values from the list of the Set property.
Type	No	No		The message name. If you are using the MRM parser and want to check that the incoming message is a particular message type, select Check type and enter the name of the message in the Type property. Leave Type clear for the other parsers.
Check type	Yes	No	Cleared	If you select the check box, the incoming message is checked against the Type property. If you are using the MRM parser and want to check that the incoming message is a particular message type, select Check type and enter the name of the message in the Type property.

The Validation properties of the Validate node are described in the following table.

If you are using the XMLNSC, MRM, or IDOC parser and want to validate the body of messages against the message set, select the required validation properties on the **Validation** tab. For more details, see "Validating messages" on page 204 and "Validation properties" on page 1445.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Validate	No	Yes	None	This property controls whether validation takes place. Valid values are None, Content and Value, Content, and Inherit.	validateMaster
Failure Action	No	No	Exception	This property controls what happens if validation fails. You can set this property only if you set Validate to Content or Content and Value. Valid values are User Trace, Local Error Log, Exception, and Exception List.	

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Warehouse node

Use the Warehouse node to interact with a database in the specified ODBC data source.

This topic contains the following sections:

Purpose

The Warehouse node is a specialized form of the Database node that stores the entire message, parts of the message, or both, in a table within the database. You define what is stored by creating mappings that use the data from the input message to identify the action that is required.

You can use the Warehouse node:

- To maintain an audit trail of messages that are passing through the broker
- For offline or batch processing of messages that have passed through the broker (data mining)
- As a source from which to reprocess selected messages in the broker

Use standard database query and mining techniques to retrieve messages that you have stored in the warehouse. (No explicit support is provided by WebSphere Message Broker.)

You must have created or identified the following items:

- Input data in the form of a message set and message
- An ODBC connection to the database
- A database and database table to store the message
- At least two columns in the table: one for the binary object (the message), and one for the timestamp

The Warehouse node is contained in the **Database** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

When you use the Warehouse node, you can store the following elements in the database that is associated with the node:

- The entire message, optionally with an associated timestamp. The message is stored as a binary object, with the timestamp in a separate column. This option has two advantages:
  - You do not need to decide beforehand how you will use the warehoused data; because you have stored it all, you can retrieve all the data and apply data mining tools to it later.
  - You do not need to define a specific database schema for every type of message that might pass through the broker. In a complex system, many different message types might be processed, and the time involved in

defining a unique schema for each message type can become prohibitive. You can precede each Warehouse node with a Compute node that converts each message into a canonical warehouse format with a common schema, or you can store the whole message as a binary object.

- Selected parts of the message, optionally with an associated timestamp, which requires a defined database schema for that message type. The message is mapped to true type so, for example, a character string in the message is stored as a character string in the database.

Terminals and properties

When you have put an instance of the Warehouse node into a message flow, you can configure it. For more information, see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view. (If you double-click a Warehouse node, you open the New Message Map dialog box.) All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

The terminals of the Warehouse node are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the input message is propagated if a failure is detected during the computation. If you have selected Treat warnings as errors, the node propagates the message to this terminal even if the processing completes successfully.
Out	The output terminal that outputs the message following the execution of the database statement.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The Warehouse node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	Warehouse	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The Warehouse node Basic properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Data source	No	Yes		The ODBC data source name of the database that contains the tables to which you refer in the mappings that are associated with this node (identified by the Field mapping property). This name identifies the appropriate database as it is known on the system on which this message flow is to run. The broker connects to this database with user ID and password information that you have specified on the mqsicreatebroker, mqsichangebroker, or mqsisetdbparms command. z/OS On z/OS systems, the broker uses the broker started task ID, or the user ID and password that are specified on the mqsisetdbparms command JCL, BIPSDBP in the customization data set <hlq>.SBIPPROC.	dataSource

Property	M	C	Default	Description	mqsiapplybaroverride command property
Field mapping	Yes	No	Warehouse	The name of the mapping routine that contains the statements that are to be executed against the database or the message tree. By default, the name that is assigned to the mapping routine is identical to the name of the mappings file in which the routine is defined, and the default name for the file is the name of the message flow concatenated with the name of the node when you include it in the message flow (for example, `MFlow1_Warehouse.msgmap` for the first Warehouse node in message flow MFlow1). You cannot specify a value that includes spaces. If you click **Browse** next to this entry field, a dialog box is displayed that lists all available mapping routines that can be accessed by this node. Select the routine that you want and click **OK**; the routine name is set in Field mapping. To work with the mapping routine that is associated with this node, double-click the node, or right-click the node and select **Open Mappings**. If the mapping routine does not exist, it is created for you with the default name in the default file. If the file exists, you can also open file *`flow_name_node_name`*`.msgmap` in the Broker Development view. The content of the mapping routine determines what is stored in the database, and in what format. You can, for example, store all or just a part of each message. You can also store the data as binary data, or store each field in the same format as it is in the message (for example, a character field in the message is stored as character in the database). A mapping routine is specific to the type of node with which it is associated; you cannot use a mapping routine that you have developed for a Warehouse node with any other node that uses mappings (for example, a DataInsert node). If you create a mapping routine, you cannot call it from any other mapping routine, although you can call it from an ESQL routine. For more information about working with mapping files, and defining their content, see “Developing message mappings” on page 546.	

Property	M	C	Default	Description	mqsiapplybaroverride command property
Transaction	Yes	No	Automatic	The transaction mode for the node. Select the value that you require: • If you select Automatic (the default), the message flow, of which the Warehouse node is a part, is committed if it is successful; that is, the actions that you define in the mappings are taken and the message continues through the message flow. If the message flow fails, it is rolled back. Therefore, selecting Automatic means that the ability to commit or roll back the action of the Warehouse node on the database depends on the success or failure of the entire message flow. • If you select Commit, any uncommitted actions that are taken in this message flow are committed on the database that is connected to this node, irrespective of the success or failure of the message flow as a whole. The changes to the database are committed even if the message flow itself fails.	
Treat warnings as errors	Yes	No	Cleared	For database warning messages to be treated as errors, and the node to propagate the output message to the Failure terminal, select Treat warnings as errors. The check box is cleared by default. When you select the check box, the node handles all positive return codes from the database as errors and generates exceptions in the same way as it does for the negative, or more serious, errors. If you do not select the check box, the node treats warnings as typical return codes, and does not raise any exceptions. The most significant warning raised is `not found`, which can be handled as a typical return code safely in most circumstances.	
Throw exception on database error	Yes	No	Selected	For the broker to generate an exception when a database error is detected, select Throw exception on database errors. The check box is selected by default. If you clear the check box, you must handle the error in the message flow to ensure the integrity of the broker and the database; the error is ignored if you do not handle it through your own processing, because you have chosen not to invoke the default error handling by the broker; for example, you can connect the Failure terminal to an error processing subroutine.	

XSLTransform node

Use the XSLTransform node to transform an XML message to another form of message, according to the rules provided by an XSL (Extensible Stylesheet Language) style sheet, and to set the Message domain, Message set, Message type, and Message format for the generated message.

This node was formerly known as the XMLTransformation node.

This topic contains the following sections:

Purpose

You can specify the location of the style sheet to apply to this transformation in three ways:

- Use the content of the XML data within the message itself to transform the message according to a style sheet that the message itself defines.
- Set a value in the LocalEnvironment folder. You must set this value in a node that precedes the XSLTransform node (for example, a Compute node). You can therefore use a variety of inputs to determine which style sheet to use for this message, such as the content of the message data, or a value in a database.
- Use node properties to ensure that the transformation that is defined by this single style sheet is applied to every message that is processed by this node.

An XSLT (Extensible Stylesheet Language for Transformations) compiler is used for the transformation if the style sheet is not embedded within the message, and the node cache level (node property Stylesheet Cache Level) is greater than zero. If the XSLT is cached, the performance is improved because the XSLT is not parsed every time it is used.

If the prologue of the input message body contains an XML encoding declaration, the XSLTransform node ignores the encoding, and always uses the CodedCharSetId in the message property folder to decode the message.

The XSLTransform node is contained in the **Transformation** drawer of the palette, and is represented in the workbench by the following icon:

Using this node in a message flow

For an example of how to use this node, consider two news organizations that exchange information on a regular basis. One might be a television station, the other a newspaper. Although the information is similar, the vocabulary that is used by the two is different. This node can transform one format to the other by applying the rules of the specified style sheet. If you specify the style sheet in the message (either the XML data or the LocalEnvironment), the same node can perform both transformations.

You cannot use relative path external entities that are defined in the DTD of input messages (for example, `<!DOCTYPE note [<!ENTITY chap1 SYSTEM "chap1.xml">]>`). Use an absolute path definition.

The XSLTransform node supports a number of local environment message tree variables, which you can use to dynamically alter the values that are set in the node's properties. For more details, see "Using local environment variables to set properties" on page 1318.

You can use style sheets in two different ways with the XSLTransform node. For more details, see "Deployed and non-deployed style sheets" on page 1319.

If you have large XML messages and receive an `out of memory` error, use the mqsireportproperties command to see the current value of the Java Heap size for the XSLT engine:

```
mqsireportproperties brokerName -e executionGroupLabel
                                 -o ComIbmJVMManager -n jvmMaxHeapSize
```

Use the mqsichangeproperties command to increase the Java Heap size:

```
mqsichangeproperties brokerName -e executionGroupLabel
                                 -o ComIbmJVMManager -n jvmMaxHeapSize -v newSize
```

In the previous examples, replace *brokerName*, *executionGroupLabel*, and *newSize* with the appropriate values.

The value that you choose for *newSize* depends on the amount of physical memory that your computer has, and how much you are using Java. A value in the range 512 MB (536870912) to 1 GB (1073741824) is typically sufficient.

Look at the following sample for more details about how to use the XSLTransform node:

- XSL Transform

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

Configuring the XSLTransform node

When you have put an instance of the XSLTransform node into a message flow, you can configure it; see "Configuring a message flow node" on page 276. The properties of the node are displayed in the Properties view.

All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

Terminals and properties

The XSLTransform node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts the message for processing by the node.
Failure	The output terminal to which the original message is routed if an error is detected during transformation.
Out	The output terminal to which the successfully transformed message is routed.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the BAR file to deploy it).

The XSLTransform node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short description	No	No		A brief description of the node.
Long description	No	No		Text that describes the purpose of the node in the message flow.

The XSLTransform node Stylesheet properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Stylesheet name	No	Yes		The name of the style sheet, which is used if the style sheet specification is searched for in node properties. To specify a style sheet by using node properties, enter the required value for Stylesheet name. Specify a principal style sheet using one of the following methods: • Click **Browse** next to Stylesheet name. The identified principal style sheet and all its relatively-referenced descendant style sheets are added automatically to the BAR file when you add a message flow to a BAR file (if both they and their parent style sheets are available). • To identify an already deployed, or ready to be deployed, style sheet, use the Stylesheet name property, and leave the Stylesheet directory property blank. • In the Message Flow editor, drag an `.xslt` file onto the XSLTransform node; the Stylesheet name is set automatically.	stylesheetName

The XSLTransform node Advanced properties are described in the following table.

Property	M	C	Default	Description	mqsiapplybaroverride command property
Stylesheet directory	No	Yes		The path where the style sheet is located. This path is used by all location methods. If the style sheet identification is fully qualified, the Stylesheet directory property is ignored; otherwise, the value that you set in this property is added to the beginning of the specification, regardless of where it is found.	stylesheetPath

Property	M	C	Default	Description	mqsiapplybaroverride command property
Stylesheet cache level	No	No	5	The number of compiled or parsed style sheets that are stored in this instance of the node. Enter a positive integer between zero (0) and 100. The default value is 5. If you set this property to zero (0), no style sheet is cached, and style sheets are interpreted rather than compiled. If your message flow does not set the style sheet dynamically by using the local environment, the same style sheet is always used, and any value greater than zero ensures that it is compiled. If your message flow does set the style sheet dynamically, you can tune the value to improve performance. If you set this property to a character other than a positive integer, a flow configuration exception error message is issued. The style sheet cache is retained for the life of the node, and is cleared when the node is deleted from the flow, when the flow is deleted, or when the execution group is stopped. If you change a cached style sheet (by redeploying or replacing the file in the file system), the XSLTransform node that is holding the cache replaces the cached version with the modified (latest) version before a new message is processed. However, if you are changing several style sheets, stop relevant message flows before you make any changes. If you do not stop the relevant message flows before you make the changes, the order of the changes cannot be guaranteed by running message flows, which might cause an incompatibility between the style sheets that are changed. Use the mqsireload command to reload a style sheet; however, this command does not prevent incompatibility. Consider performance when you set this property. Typically, when you set this property to a number greater than the default value of 5, performance is faster because style sheet re-compilation is less likely. However, cached style sheets use Java virtual machine (JVM) heap space; if you keep too many cached style sheets, the JVM is likely to slow. In addition, if the cached style sheets are not used regularly and you set this property to a large number, performance can be affected because compiling style sheets increases processor usage. Therefore, to decide on the most suitable value for this property, you must balance how many style sheets you use with the size of the style sheets, the size of the JVM heap space, and your usage pattern. For more information about the JVM heap space, see “JVM heap sizing” on page 201.	

The XSLTransform node Output Message Parsing properties are described in the following table.

Property	M	C	Default	Description
Message domain	No	No	BLOB	The message domain that is associated with the output message. To associate a specific parser with the output message, specify the new domain in Message domain. The default value is BLOB. This domain is applied to the output message. Alternatively, use Inherit to associate the parser that owned the input message. XML is deprecated; use XMLNSC instead. You can also specify a user-defined parser if appropriate.
Message set	No	No		The message set that is associated with the output message. If you are using the MRM parser, or the XMLNSC parser in validating mode, select the Message set that you want to use. This list is populated with available message sets when you select MRM or XMLNSC as the domain. If you set this property, then subsequently update the project dependencies to remove this message set reference, a warning is issued. Either update the Message set property, or restore the reference to this message set project.
Message type	No	No		The message type that is associated with the output message. If you are using the MRM parser, select the correct message from the list in Message type. This list is populated with messages that are defined in the Message set that you have selected.
Message format	No	No		The message format that is associated with the output message. If you are using the MRM parser, select the XML physical format for the output message from the list in Message format. This list includes all the physical formats that you have defined for this Message set.
Character set	No	No		The numeric value of the character set for the output message. To specify a character set for the output message by using node properties, specify the required value for Character set. The value that you specify must be numeric; for example, specify 1200 to encode the output message as UTF-16.

The XSLTransform node Parser Options are described in the following table.

Property	M	C	Default	Description
Parse timing	No	No	On Demand	This property controls when an output message is parsed. Valid values are On Demand, Immediate, and Complete. Parse timing is, by default, set to On Demand, which causes parsing of the message to be delayed. To cause the message to be parsed immediately, see “Parsing on demand” on page 1449.
Build tree using XML schema data types	No	No	Cleared	This property controls whether the XMLNSC parser creates syntax elements in the message tree with data types taken from the XML Schema. You can select this property only if you set the Validate property to Content or Content and Value. For more information, see “Manipulating messages in the XMLNSC domain” on page 408.
Use XMLNSC compact parser for XMLNS domain	No	No	No	This property controls whether the XMLNSC Compact Parser is used for output messages in the XMLNS Domain. If you set this property, the message data appears under XMLNSC in nodes that are connected to the output terminal when the input MQRFH2 header or Domain is XMLNS.

The XSLTransform node Validation properties are described in the following table. Set the validation properties for the parser to validate the body of messages against the Message set. (If a message is propagated to the Failure terminal of the node, it is not validated.) For more details, see “Validating messages” on page 204 and “Validation properties” on page 1445.

Property	M	C	Default	Description
Validate	No	Yes	None	This property controls whether validation takes place of the output message. Valid values are None, Content, Content and Value, and Inherit.
Failure action	No	No	Exception	This property controls what happens if validation of the output message fails. You can set this property only if you set Validate to Content and Value or Content. Valid values are User Trace, Local Error Log, Exception, and Exception List.

The XSLTransform node Detail Trace properties are described in the following table.

Property	M	C	Default	Description
Trace setting	Yes	No	Off	This property is deprecated. Start user trace instead. The user trace contains the same XSL trace information. If you set this property, it does not affect user trace. In previous versions of WebSphere Message Broker, this property controls whether tracing is on or off. If tracing is on, low level tracing is recorded in a file.

The Monitoring properties of the node are described in the following table.

Property	M	C	Default	Description
Events	No	No	None	Events that you have defined for the node are displayed on this tab. By default, no monitoring events are defined on any node in a message flow. Use **Add**, **Edit**, and **Delete** to create, change or delete monitoring events for the node; see "Configuring monitoring event sources using monitoring properties" on page 146 for details. You can enable and disable events that are shown here by selecting or clearing the **Enabled** check box.

Adding keywords to XSL style sheets

Embedded keywords in an XSL style sheet; their location is not restricted. You can also add a keyword as an XML comment.

XML comments must have the following format:

```
$MQSI keyword = value MQSI$
```

The example shows how to add the keyword of **author** with the value John to an XML style sheet:

```
<?xml version="1.0" encoding="UTF-8">
<!-- $MQSI author = John MQSI$ -->
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text" encoding="UTF-8"/>
<xsl:template match="/">
<xsl:value-of select="message"/>
</xsl:template>
</xsl:stylesheet>
```

The Configuration Manager does not extract `version="1.0"` from this example, because the value is not bounded by the `$MQSI` and `MQSI$` keywords.

Restrictions within keywords

Do not use the following characters within keywords, because they might cause unpredictable behavior:

```
^ $ . | \ < > ? + * = & [ ] ( )
```

You can use these characters in the values that are associated with keywords; for example:

- `$MQSI RCSVER=$id$ MQSI$` is acceptable
- `$MQSI $name=Fred MQSI$` is not acceptable

Using local environment variables to set properties

The XSLTransform node supports a number of local environment message tree variables, which you can use to dynamically alter the values that are set in the node's properties.

The following table lists each local environment variable against the name of the node property that it overrides:

Local environment variable name	Node property name
XSL.StyleSheetName	Stylesheet name
XSL.MessageDomain	Message domain
XSL.MessageSet	Message set
XSL.MessageType	Message type
XSL.MessageFormat	Message format
XSL.OutputCharSet	Character set

This node searches for the name of the style sheet to be used by interrogating, in the following order:

1. The input message.

   The node searches the message XML data for information about the location of the style sheet. For example, the XML data might contain:

   ```
   <?xml-stylesheet type="text/xsl" href="me.xsl"?>
   ```

   and `"me.xsl"` is then used as the name of the style sheet.
2. The local environment.

   If no style sheet name is found in the input message, the node searches the local environment that is associated with the current message for style sheet information stored in an element called XSL.StyleSheetName.

   This node was available in Version 5.0 and Version 6.0, and element ComIbmXslXmltStylesheetname was used for the name of the style sheet, therefore the current node checks both elements. If both are present, the value in XSL.StyleSheetName takes precedence.
3. The node properties.

   If no style sheet name is found in the input message or local environment, the node uses the Stylesheet name and Stylesheet directory properties to determine the correct values.

The node searches for the message domain, message set, message type, and message format to use for the output message by interrogating, in the following order:

1. The local environment.

   The node searches the local environment that is associated with the current message for message domain, message set, message type, and message format information stored in elements called XSL.MessageDomain, XSL.MessageSet, XSL.MessageType, and XSL.MessageFormat.
2. The node's properties.

   If no message domain, message set, message type, or message format information is found in these local environment variables, the node uses the Message domain, Message set, Message type, and Message format properties to determine the correct values.

If the node cannot determine the message domain from either XSL.MessageDomain or the Message domain property, the default value of BLOB is used. No default values exist for message set, message type, and message format.

The node searches for the character set to use for the output message by interrogating, in the following order:

1. The local environment.

   The node searches the local environment that is associated with the current message for character set information stored in an element called XSL.OutputCharSet; for example, to encode the output of the transformation as UTF-8, enter the value 1208 as a string in this element.

   This node was available in Version 5.0 and Version 6.0, and element ComIbmXslXmltOutputcharset was used for the output character set, therefore the current node checks both elements. If both are present, the value in XSL.OutputCharSet takes precedence.
2. The node's properties.

   If no character set information is found in the local environment, the node uses the Character set property to determine the correct value.

   If you set a value for Character set, the value that you enter must be numeric; for example, to encode the output of the transformation as UTF-16, enter 1200.

If the node cannot determine the output character set from either of these two sources, because either no value is set or the selection priorities are set to zero, the default value of 1208 (UTF-8) is used.

Be aware of the following factors if the input to the XSLTransform node is generated from the XMLNSC parser or the MRM parser. The XMLNSC parser discards certain information in XML documents, such as processing instructions and comments, if you do not set properties to retain this information in a preceding node. To ensure that the XSLTransform node transforms the message correctly, set the Retain mixed content, Retain comments, and Retain processing instructions properties correctly on the preceding node (for example, an MQInput node). The MRM parser also discards this information, but you cannot retain information for this parser, therefore avoid using the MRM domain if such information is vital to your transformation.

Deployed and non-deployed style sheets

You can use style sheets in two different ways with the XSLTransform node.

Deployed style sheets

Deployed style sheets are style sheets that you import into a broker archive (BAR) file and deploy to target systems. Deployed style sheets are managed by the broker. A *principal style sheet* is the root style sheet that is referenced in a message flow; for example, a reference to a principal style

sheet in the Eclipse workspace, `C:\\project1\a\b.xsl` must be specified as `a/b.xsl` (or `./a/b.xsl`). A principal style sheet can reference (include or import) its descendent style sheets.

Non-deployed style sheets
Non-deployed style sheets are style sheets that you store in a location where the XSLTransform node can access them. Non-deployed style sheets are not managed by the broker.

For more information, see Migrating style sheets and XML files from Version 5.0.

Deployment of deployed style sheets or XML files

Before you can configure the XSLTransform node, you must understand how to work with style sheets. A style sheet can refer to both another XML file and a style sheet. To use deployed style sheets or XML files:

1. Make sure that the files have the correct file name extensions.

   Style sheets that are to be deployed must have either `.xsl` or `.xslt` as their file extension, and XML files to be deployed must have `.xml` as their file extension.

2. Import the files into the Eclipse workspace.

   Import into an Eclipse workspace project all style sheets and XML files that are to be deployed. Put location-dependent descendant style sheets, or XML files that are to be deployed, in the correct directory structure relative to their parent style sheets. Do not put in the Eclipse workspace location-dependent descendants that you do not want to deploy.

3. Make sure that all references to the files are relative.

   Typically, all references to a deployed style sheet must be made relative, no matter where they are displayed. A reference to a principal style sheet must be made relative to the root of the relevant Eclipse workspace project.

   The only exception is when you specify a principal style sheet as the Stylesheet name property on an XSLTransform node; you can use an absolute path that points to the correct directory structure in the Eclipse workspace. If the principal style sheet is found, the system resets the node property automatically to the correct relative value.

   The system also performs an automatic deployment of the principal style sheet, together with all of its location-dependent descendant style sheets that are available in the relevant Eclipse workspace project. All references to the location-dependent descendant style sheets (or XML files) of a principal style sheet must be made relative to the location of their parent style sheets. For example, if style sheet `//project1/a/b.xsl` references style sheet `//project1/a/c/d.xsl`, the reference must be changed to `c/d.xsl` (or `./c/d.xsl`).

4. Handle non-deployed child style sheets or XML files.

   Style sheets can refer to other style sheets. If you have a relatively-referenced child style sheet (or XML file) that is not to be deployed, yet its parent is, make sure that the child style sheet is placed in the correct location under *workpath*`/XSL/external` (*workpath*`/XML/external`), where *workpath* is the full path to the working directory of the broker. You can use the `MQSI_WORKPATH` environment variable to find the location of the workpath on your system; for example, on Windows XP systems, the default workpath is `MQSI_WORKPATH=C:\Documents and Settings\All Users\Application Data\IBM\MQSI`.

   A broker automatically associates the execution group deployed storage tree, *workpath*`/`**`XSL`**`/external`, and *workpath*`/`**`XML`**`/external` tree, together. Therefore if,

for example, the document `b/c.xml` is not found in the broker's deployed storage, the broker automatically searches for a reference to it in the *workpath*`/XML/external/a/b` directory in the deployed principal style sheet `a/style.xsl`. Relative path references must also be used for files that have been deployed but which are not available in the workspace.

5. Deploy the files.

   Deploy manually only those style sheets or XML files that are not picked up by the system (the Message Broker Toolkit provides warnings about these files). If you click **Browse** for the node, or provide the full path of the location of the style sheet in the Eclipse workspace, the style sheet is included automatically in the BAR file.

   To deploy manually, add the files to be deployed to a broker archive. For more information, see Adding files to a broker archive and "Adding keywords to XSL style sheets" on page 1317.

   For every execution group that uses the XSLTransform node, perform one of the following actions:

   - Include the style sheet in the *workpath*`/XSL/external` directory on the broker; do not include the style sheet in the BAR file.

     If a style sheet in the *workpath*`/XSL/external` directory shares the same path and name with a deployed style sheet, the deployed style sheet is used.
   - Include the style sheet in the BAR file and deploy the BAR file. If multiple BAR files include the same style sheet name, the style sheet from the last BAR file that was deployed is used.
   - Deploy the style sheet in its own BAR file. If the BAR files use XSLTransform nodes, but do not include the style sheet, the Message Broker Toolkit issues warning messages.

Description properties for a message flow

The description properties for a message flow include the Version, Short Description and Long Description. To view and edit the properties of a message flow click **Flow** → **Properties**.

Property	Type	Meaning
Version	String	You can enter a version for the message flow in this field. This allows the version of the message flow to be displayed using the Eclipse properties view. A default for this field can be set in the messages flow preferences.
Short Description	String	You can enter a short description of the message flow in this field.
Long Description	String	You can add information to enhance the understanding of the message flow's function in this field. It is a string field and any standard alphanumeric characters can be used. You can also use this field to define a keyword and its value that will display for the deployed message flow in the properties view of Eclipse. An example is: `$MQSI Author=Fred MQSI$` When the properties of the deployed message flow are displayed, this will add a row to the display showing "Author" as the property name and "Fred" as its value. For information on keywords see "Guidance for defining keywords" on page 1322.

Guidance for defining keywords

You can add extra information to an object in the form of one or more keywords.

This information can display information about an object after the object has been deployed. The default information that is displayed is the time the object was deployed and the last time the object was modified.

You can define custom keywords, and their values that the workbench interprets as additional information to be displayed, in the properties view. For example, you can define keywords for "Author" and "Subflow 1 Version":

```
$MQSI Author=John Smith MQSI$
$MQSI Subflow 1 Version=v1.3.2 MQSI$
```

The following table contains the information that is displayed by the workbench:

Object name	
Deployment Time	28-Aug-2009 15:04
Modification Time	28-Aug-2009 14:27
Version	v1.0
Author	John Smith
Subflow 1 Version	v1.3.2

In this display, the version information has been defined by using the **Version** property of the object. If the version information had not been defined, it would be omitted from this display.

Use the following syntax to define a keyword and its associated value:

```
$MQSI KeywordName = KeywordValue MQSI$
```

Where:

$MQSI
: $MQSI opens the definition. It can be followed by an optional underscore or white-space character that is ignored.

KeywordName
: The name of the keyword for which you are setting the value. It is made up of a sequence of alphanumeric characters apart from the equals (=) sign. It can contain white-space characters, but leading or trailing white-space characters are omitted.

=
: The equals (=) sign is the delimiter between the keyword and the value that you are setting it to.

KeywordValue
: The value to which the keyword is set. It is made up of a sequence of alphanumeric characters. It can contain white-space characters, but leading or trailing white-space characters are omitted.

MQSI$
: MQSI$ closes the keyword definition.

Examples

Example definitions	Interpreted keyword and value	Comments
$MQSIAuthor=JohnMQSI$ or $MQSI Author=John MQSI$ or $MQSI Author = John MQSI$	Keyword = "Author" Value = "John"	Each of these examples shows what can be set and that the leading and trailing white-space characters for the name and value parameters are ignored.
$MQSI_Author = John MQSI$	Keyword = "Author" Value = "John"	The first character after $MQSI can be an underscore character. The underscore character is omitted in the interpreted keyword. If a second underscore character appears, this forms part of the keyword name.
$MQSI Flow designer = John Smith MQSI$	Keyword = "Flow designer" Value = "John Smith"	White-space characters are accepted for each parameter value.
$MQSI bar = MQSI$	Keyword = "bar" Value = ""	The keyword value can be set to an empty ("") string.
$MQSI_mqsitag=$MQSI$MQSI$	Keyword = "mqsitag" Value = "$"	This example is a poorly formatted definition. After defining the keyword name, the parser is looking to find the delimiters that form the boundary of the value to be set. In this case, the only character before the MQSI$ that closes the definition is a '$', and that is set as the keyword value.
$MQSI=barMQSI$		This pattern is ignored because the keyword name cannot be an empty string.
$MQSItagbarMQSI$		This pattern is ignored because there is not a separator (=) between the keyword name and the keyword value.

Do not use the following keywords:

VERSION
When you use the Message Broker Toolkit to edit message flows and dictionaries, it is possible to set the **Version** property in the Properties pane, which you can then view in the Broker Archive file editor. If you set this property, a keyword called VERSION is added to the resulting `.cmf` or dictionary file. For this reason, do not add `$MQSI_VERSION=...MQSI$` to these files.

BAR The BAR keyword is associated with each object automatically when it is deployed and it contains the full path name of the broker archive file that deployed the object.

The values of both keywords are defined programmatically in the class `com.ibm.broker.config.proxy.DeployedObject`.

Restrictions within keywords

Do not use the following characters within keywords because they cause unpredictable behavior:

```
^ $ . | \ < > ? + * = & [ ] ( )
```

You can use these characters in the values that are associated with keywords; for example:

- `$MQSI RCSVER=$id$ MQSI$` is acceptable
- `$MQSI $name=Fred MQSI$` is not acceptable

Configurable message flow properties

When you add a message flow to a broker archive (BAR) file in preparation for deploying it to a broker, you can set additional properties that influence its run time operation. These properties are available for review and update when you select the **Manage and Configure** tab for the broker archive file.

Additional Instances

Specifies the number of additional threads that the broker can use to service the message flow. These additional threads are created only if there are sufficient input messages. You can use up to 256 threads. The default value is 0. Additional threads can increase the throughput of a message flow but you must consider the potential effect on message order.

If the message flow processes WebSphere MQ messages, you can configure the message flow to control the message order. Set the Order Mode property on the MQInput node accordingly. You might also need to set the Commit by Message Group and Logical Order properties.

An MQInput node opens the input queue with MQOO_INPUT_AS_Q_DEF, which uses the DEFSOPT property of the input queue. Therefore, you must ensure that the input queue has been defined with DEFSOPT(SHARED) and with the SHARE property set to enable multiple broker threads to read from the input queue. If these properties are not set in this way, the message flow threads report that the queue is in use (MQRC=2042), and the message flow might stop processing messages on the input queue.

If you have multiple input nodes in your message flow, the available additional threads might not be allocated evenly between the different input nodes. In an extreme case, all the threads might be allocated to a single input node, and only one aspect of message flow throughput is improved. To avoid this problem, you can use the Additional Instances Pool property, together with the Additional Instances property, to allocate a pool of additional instance threads for each input node.

Additional Instances Pool

Specifies whether additional instance threads for an input node are allocated from a thread pool for the whole message flow, or from a thread pool for use by that node only. The value of the Additional Instances property that is specified for the node controls the number of instances in the pool.

If your message flow contains multiple input nodes, use the Additional Instances Pool and Additional Instances properties to ensure that each of the input nodes is allocated the required number of additional instances. Setting these properties allows you to fine-tune your message flow operation, therefore if you know that one input node will receive twice as many input messages as another node, you can give it twice the number of additional threads.

- To request additional instance allocation from the message flow thread pool, set Additional Instances Pool to Use pool associated with message flow.
- To request additional instance allocation from thread pool allocated to the node, set Additional Instances Pool to Use pool associated with node.

If the Additional Instances Pool property is not available for a node, the behavior is the same as if Additional Instances Pool is set to Use pool associated with message flow. You can set this property on the MQInput node, FileInput node, WebSphere Adapters input nodes, and Web Services input nodes.

Commit Count

For WebSphere MQ messages, specifies how many input messages are processed by a message flow before a sync point is taken (by issuing an MQCMIT).

The default value of 1 is also the minimum permitted value. Change this property to avoid frequent MQCMIT calls when messages are being processed quickly and the lack of an immediate commit can be tolerated by the receiving application.

Use the Commit Interval to ensure that a commit is performed periodically when not enough messages are received to fulfill the Commit Count.

This property has no effect if the message flow does not process WebSphere MQ messages.

Commit Interval

For WebSphere MQ messages, specifies a time interval at which a commit is taken when the Commit Count property is greater than 1 (that is, where the message flow is batching messages), but the number of messages processed has not reached the value of the Commit Count property. It ensures that a commit is performed periodically when not enough messages are received to fulfill the Commit Count.

The time interval is specified in seconds, as a decimal number with a maximum of three decimal places (millisecond granularity). The value must be in the range 0.000 through 60.000. The default value is 0.

This property has no effect if the message flow does not process WebSphere MQ messages.

Coordinated Transaction

Controls whether the message flow is processed as a global transaction, coordinated by WebSphere MQ. Such a message flow is said to be fully globally coordinated. The default value is No.

Use coordinated transactions only where you need to process the message and any database updates that are performed by the message flow in a single unit-of-work, using a two-phase commit protocol. In this case, both the message is read and the database updates are performed, or neither is done.

If you change this value, ensure that the queue manager for the broker is configured correctly. If you do not set up the queue manager correctly, the broker generates a message when the message flow receives a message to indicate that, although the message flow is to be globally coordinated, the queue manager configuration does not support coordination.

See Supported databases for information about which databases are supported as participants in a global transaction, and the *System Administration* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online for how to configure WebSphere MQ and the database managers.

This property has no effect if the message flow does not process WebSphere MQ messages.

User-defined properties

The initial value of a user-defined property (UDP) can be modified at design time by the Message Flow editor, or overridden at deployment time by the Broker Archive editor. The advantage of UDPs is that their values can be changed by operational staff at deployment time. If, for example, you use UDPs to hold configuration data, you can configure a message flow for a particular computer, task, or environment at deployment time, without having to change the code at the node level. You can also query and set the values of user-defined properties at run time by using the Configuration Manager Proxy (CMP). For example, a systems monitoring tool might use the CMP API to modify the value of a user-defined property at run time to change the behavior of the message flow.

For introductory information about UDPs and dynamic UDPs, see "User-defined properties in ESQL" on page 305 and "User-defined properties" on page 134.

For information about configuring UDPs at deployment time, see "Configuring a message flow at deployment time with user-defined properties" on page 454.

For information about configuring UDPs at run time, see Setting user-defined properties at run time in a CMP application.

You can view and update other configurable properties for the message flow. The properties that are displayed depend on the nodes within the message flow; some have no configurable properties to display. The node properties that are configurable are predominantly system-related properties that are likely to change for each broker to which the message flow is deployed. These properties include data source names and the names of WebSphere MQ queues and queue managers. For full details of configurable properties for a node, see the appropriate node description.

WebSphere Adapters properties

Reference information about the properties that you set for WebSphere Adapters nodes.

See the properties for the Enterprise Information System (EIS) to which you want to connect:

- "WebSphere Adapter for SAP properties"
- "WebSphere Adapter for Siebel properties" on page 1403
- "WebSphere Adapter for PeopleSoft properties" on page 1425

WebSphere Adapter for SAP properties

Reference information to which to refer when you connect to an SAP application.

- "Business object information (SAP)" on page 1327

- "Configuration properties for the WebSphere Adapter for SAP Software" on page 1334

Business object information (SAP)

A business object contains application-specific information (metadata) about how the adapter processes the business object, and about the operation to be performed on the business object.

The name of the business object is generated by the Adapter Connection wizard in accordance with the naming convention for the adapter.

For more information, see the following topics:

- "Supported data operations (SAP)"
- "Naming conventions" on page 1330

Supported data operations (SAP):

For outbound processing, an operation is the name of the action that is implemented by the adapter so that the message flow can perform the operation on the SAP server.

The adapter uses the application-specific information (ASI) inside the business object definition to implement the operation. The name of the operation typically indicates the type of action to be implemented, such as create or update. For inbound processing, adapters implement an operation by delivering events to their endpoints. For inbound processing, the action that is associated with the event varies depending on the interface (ALE or Advanced event processing). When the interface is ALE, the action is pushed to the adapter and the adapter delivers the event to an endpoint. When the interface is Advanced event processing, the event status is polled by the adapter and processed accordingly.

For more information, see the following topics:

- "Supported data operations on BAPI business objects"
- "Supported data operations on ALE business objects" on page 1328
- "Supported data operations of Query interface for SAP Software business objects" on page 1329
- "Supported data operations on Advanced event processing business objects" on page 1329

Supported data operations on BAPI business objects:

The operations that are supported by BAPI business objects are the same as those supported by BAPI work units. BAPI result sets support only the RetrieveAll operation.

For BAPI outbound processing, the operation of a BAPI business object is the name of the BAPI call that an adapter issues on the SAP server. The BAPI method determines the operation that is associated with it. The adapter uses the application-specific information (ASI) inside the business object definition to implement the operation. Operations of a business object are called by the component that makes calls to SAP through the adapter. The SAP JCo APIs are used to make the call to the SAP system.

BAPIs and BAPI unit of work

The following table defines operations that the adapter supports for BAPIs and BAPI work units. The definitions in the table are the expected uses for the operations. The action that is taken in the SAP application is based on the meaning of the BAPI itself.

Table 24. Supported operations: BAPI business objects

Operation	Definition
Create	The top-level business object and all contained children are created.
Update	The top-level business object is modified. This operation can include adding and deleting child objects.
Delete	The top-level business object and any contained children are deleted.
Retrieve	The top-level business object and any contained children are retrieved.

For an operation that is not supported, the adapter logs the appropriate error and produces a ResourceException.

Result sets

The following table defines the operation that the adapter supports for BAPI result sets.

Table 25. Supported operation: BAPI result sets

Operation	Definition
RetrieveAll	All the matching records for the BAPI result set are retrieved.

Supported data operations on ALE business objects:

The operations that are supported by ALE business objects vary, depending on whether the business object is an outbound or inbound object. The adapter uses the application-specific information (ASI) inside the business object definition to implement the operation.

Business objects that are generated with the ALE pass-through IDoc interface are not associated with an operation.

Outbound business objects

The operation of an ALE outbound business object is called by the application component that makes calls to SAP through the adapter. The adapter supports the following outbound operation:

Table 26. Supported operation: ALE outbound business objects

Operation	Definition
Execute	This operation posts the IDoc business object to the SAP application. This operation is one-way and asynchronous. For the `CWYAP_SAPAdapter_TX.rar` version of the adapter, the transaction ID is returned.

Inbound business objects

For ALE inbound business objects, the application-specific information of an operation contains the message type, message code, and message function for an IDoc type. The adapter supports the following inbound operations:

Table 27. Supported operations: ALE inbound business objects

Operation	Definition
Create	The top-level business object and all contained children are created.
Update	The top-level business object is modified. This operation can include adding and deleting child objects.
Delete	The top-level business object and any contained children are deleted.

The adapter uses the IDoc control record field data to determine the operation that is set on the business object before sending it to the endpoint. The following fields in the control record are used to determine the operation:

- Logical_message_type (MESTYP)
- Logical_message_code (MESCOD)
- Logical_message_function(MESFCT)

Supported data operations of Query interface for SAP Software business objects:

The SAP Query interface supports the RetrieveAll operation, with which you can have the results of an SAP table returned to you, and the Exists operation, which you use to determine whether data can be found in the SAP table. The adapter uses the application-specific information (ASI) inside the business object definition to implement the operation.

The supported operations for the Query interface for SAP Software are listed in the following table.

Table 28. Supported operations: Query interface for SAP Software business objects

Operation	Description
RetrieveAll	This operation returns a result set in the form of a container of SAP query business objects, which represent the data for each row that is retrieved from the table. If a table business object is sent to the SAP server (instead of a container business object), the rows are returned one at a time.
Exists	This operation provides a means to check for the existence of any records in SAP for defined search criteria. The Exists operation does not return any data; it indicates whether the data exists in SAP. If no data is found, the adapter generates an exception.

Supported data operations on Advanced event processing business objects:

The operations that are supported by Advanced event processing business objects vary, depending on whether the business object is an outbound or inbound object. The adapter uses the application-specific information (ASI) inside the business object definition to implement the operation.

Outbound business objects

The operation of an Advanced event processing outbound business object is called by the message flow that makes calls to SAP through the adapter. The adapter supports the following outbound operation.

Table 29. Supported operation: Advanced event processing outbound business objects

Operation	Definition
Create	The top-level business object and all contained children are created.
Update	The top-level business object is modified. This operation can include adding and deleting child objects.
Delete	The top-level business object and any contained children are deleted.
Retrieve	The top-level business object and any contained children are retrieved.

Inbound business objects

For Advanced event processing inbound business objects, the application-specific information of an operation contains the message type, message code, and message function for an IDoc type. The adapter supports the following inbound operations.

Table 30. Supported operations: Advanced event processing inbound business objects

Operation	Definition
Create	The top-level business object and all contained children are created.
Update	The top-level business object is modified. This operation can include adding and deleting child objects.
Delete	The top-level business object and any contained children are deleted.

Naming conventions:

When the Adapter Connection wizard generates a business object, it provides a name for the business object that is based on the name of the corresponding business function in the SAP server.

The convention that is applied by the SAP server when naming a business object varies depending on whether the name is for a BAPI business object, an ALE business object, an Advanced event processing business object, or a Query interface for SAP Software business object.

For more information, see the following topics:

- "Naming conventions for BAPI business objects"
- "Naming conventions for ALE business objects" on page 1332
- "Naming conventions for Query interface for SAP Software business objects" on page 1333
- "Naming conventions for Advanced event processing business objects" on page 1333

Naming conventions for BAPI business objects:

The Adapter Connection wizard provides names for the business objects for BAPIs, BAPI work units, and BAPI result sets. The business object name reflects the structure of the business function on the SAP server.

BAPIs

When it names business objects for BAPIs, the Adapter Connection wizard adds the prefix `Sap`. The wizard also converts the name of the business function to mixed case, removing any separators such as spaces or underscores, capitalizes the first letter of each word, and it can add an element-specific suffix (for example, `Wrapper` for a top-level business object).

The following table describes the convention that is applied by the Adapter Connection wizard when it names BAPI business objects.

Table 31. Naming conventions for BAPI business objects

Element	Naming convention
Name of the top-level business object	`Sap` + *Name of the wrapper object that you specify in the Adapter Connection wizard* + `Wrapper` For example: `SapSalesOrderWrapper`
Name of the BAPI business object	`Sap` + *Name of the BAPI interface* For example: `SapBapiSalesOrderCreateFromDat1` The top-level object can contain more than one BAPI object.
Name of the child object	`Sap` + *Name of the Structure/Table* For example: `SapReturn`

If structures with the same name exist in different BAPIs, or exist in a BAPI (for example, one at the export level and one at the table level), the Adapter Connection wizard adds a unique suffix to differentiate the structures. The first structure is assigned a name (for example, `SapReturn`) and the second structure is assigned a name such as `SapReturn619647890`, where 619647890 is the unique identifier that is appended to the name by the wizard.

BAPI work units

The following table describes the convention that is applied by the Adapter Connection wizard when it names a BAPI work unit business object.

Table 32. Naming conventions for BAPI unit of work business objects

Element	Naming convention
Name of the top-level business object	`Sap` + *Name of the wrapper object that you specify in the Adapter Connection wizard* + `Txn` For example: `SapCustomerTxn`
Name of the BAPI business object	`Sap` + *Name of the BAPI interface* For example: `SapCustomer`
Name of the child object	`Sap` + *Name of the Structure/Table* For example: `SapReturn`

If structures with the same name exist in different BAPIs, or exist in a BAPI (for example, one at the export level and one at the table level), the Adapter Connection wizard adds a unique suffix to differentiate the structures. The first structure is assigned a name (for example, SapReturn) and the second structure is assigned a name such as SapReturn619647890, where 619647890 is the unique identifier that is appended to the name by the wizard.

BAPI result sets

The following table describes the convention that is applied by the Adapter Connection wizard when it names a BAPI result-sets business object.

Table 33. Naming conventions for BAPI result sets

Element	Naming convention
Name of the result set BAPI business object	Sap + *Name of the BAPI interface* For example: SapBapiCustomerGetDetail
Name of the child object	Sap + *Name of the Structure/Table* For example: SapReturn
Name of the query business object	Sap + *Formatted name of the query BAPI interface* For example: SapBapiCustomerGetList

If structures with the same name exist in different BAPIs, or exist in a BAPI (for example, one at the export level and one at the table level), the Adapter Connection wizard adds a unique suffix to differentiate the structures. The first structure is assigned a name (for example, SapReturn) and the second structure is assigned a name such as SapReturn619647890, where 619647890 is the unique identifier that is appended to the name by the wizard.

Naming conventions for ALE business objects:

The Adapter Connection wizard provides names for the ALE business graph, top-level business object, and the business object. The business object name reflects the structure of the business function on the SAP server.

If you are using the ALE pass-through IDoc interface, the following naming conventions apply:

- When you select **Generic IDoc** from the Object Discovery and Selection window, the Adapter Connection wizard creates a business object named SapGenericIDocObject. The naming convention described in the following sections does not apply to generic IDocs.
- When you discover an IDoc from the system or from a file, the object is named according to the naming convention for top-level wrapper objects, as described in Table 34 on page 1333. No other objects are generated.

When it names business objects for ALE, the Adapter Connection wizard adds a prefix of Sap. The wizard also converts the name of the IDoc and extension to mixed case, removing any separators such as spaces or underscores, capitalizes the first letter of each word, and it can add an element-specific suffix (for example, BG for business graph).

The following table describes the conventions that are applied by the Adapter Connection wizard when it names ALE business objects. The *[Name of Extension*

type IDoc] in the Naming convention column represents an optional entry. It is included in the name only if the selected IDoc is an Extension Type IDoc.

Table 34. Naming conventions for ALE business objects

Element	Naming convention
Name of the top-level wrapper object	`Sap` + *`Name of IDoc`* + *`[Name of Extension type IDoc]`* For example: `SapAlereq01`
Name of the IDoc business object for basic IDocs	`Sap` + *`Name of IDoc`* + *`BO`* For example, the business object for the IDoc MATMAS03 is: `SapMatmas03BO`
Name of the IDoc business object for extension type IDocs	`Sap` + *`Name of IDoc`* + *`Name of Extension type IDoc`* For example, the business object for the IDoc DELVRY03 and extension SD_DESADV_PDC is: `SapDelvry03SdDesadvPdc`

For an IDoc duplicate name, the Adapter Connection wizard adds a unique suffix to differentiate the business object. If an IDoc packet has two segments with the same name (for example, segOrder), the first business object is assigned the name `SapSegOrder` and the second business object is assigned a name such as `SapSegOrder619647890`, where 619647890 is the unique identifier that the Adapter Connection wizard appends to the name.

Naming conventions for Query interface for SAP Software business objects:

The Adapter Connection wizard provides names for the Query interface for SAP Software container, business graph, top-level business object, table object, and query object. The business object name reflects the structure of the business function on the SAP server

When it names business objects for the Query interface for SAP Software, the Adapter Connection wizard adds the prefix `Sap`. The wizard also converts the name of the business function or SAP table to mixed case, removing any separators such as spaces or underscores, capitalizes the first letter of each word, and it can add an element-specific suffix (for example, `Container` for a container).

The following table describes the convention that is applied by the Adapter Connection wizard when it names a Query interface for SAP Software business object.

Table 35. Naming convention for a Query interface for SAP Software business object

Element	Naming convention
Name of the container	`Sap` + *Name of the object that you specify in the wizard* + `Container` For example: `SapCustomerContainer`
Name of the table object	`Sap` + *Name of the SAP table* For example: `SapKna1`
Name of the query object	`Sap` + *Name of the SAP table* + `Querybo` For example: `SapKna1Querybo`

Naming conventions for Advanced event processing business objects:

The Adapter Connection wizard provides names for the Advanced event processing top-level business object, and the business object. The name of the business object reflects the structure of the business function on the SAP server.

When it names business objects for the Advanced event processing interface, the Adapter Connection wizard adds a prefix of `Sap`. The wizard also converts the name of the IDoc and extension to mixed case, removing any separators such as spaces or underscores, capitalizes the first letter of each word, and it might add an element-specific suffix.

The following table describes the convention that is applied by the Adapter Connection wizard when it names Advanced event processing business objects. The *[Name of Extension type IDoc]* in the Naming convention column represents an optional entry; it is included in the name only if the selected IDoc is an Extension Type IDoc.

Table 36. Naming convention for advanced event processing business objects

Element	Naming convention
Name of the top-level wrapper object	`Sap` + *`Name of IDoc`* + *`[Name of Extension type IDoc]`* For example: `SapAepreq01`
Name of the IDoc business object for basic IDocs	`Sap` + *`Name of IDoc`* + *`BO`* For example, the business object for the IDoc MATMAS03 is: `SapMatmas03BO`
Name of the IDoc business object for extension type IDocs	`Sap` + *`Name of IDoc`* + *`Name of Extension type IDoc`* For example, the business object for the IDoc DELVRY03 and extension SD_DESADV_PDC is: `SapDelvry03SdDesadvPdc`

For an IDoc duplicate name, the Adapter Connection wizard adds a unique suffix to differentiate the business object. If an IDoc packet has two segments with the same name (for example, segOrder), the first business object is assigned the name `SapSegOrder` and the second business object is assigned a name such as `SapSegOrder619647890`, where 619647890 is the unique identifier that is appended to the name by the Adapter Connection wizard.

Configuration properties for the WebSphere Adapter for SAP Software

The WebSphere Adapter for SAP Software has several categories of configuration properties, which you set with the Adapter Connection wizard when you generate or create objects and services.

You can change the resource adapter, managed connection factory, and activation specification properties in WebSphere Message Broker.

For more information, see the following topics:

- "SAP connection properties for the Adapter Connection wizard" on page 1335
- "Resource adapter properties (SAP)" on page 1344
- "Managed connection factory properties (SAP)" on page 1344
- "Activation specification properties for ALE inbound processing" on page 1369
- "Activation specification properties for Advanced event processing" on page 1386
- "Interaction specification properties (SAP)" on page 1399

SAP connection properties for the Adapter Connection wizard:

Connection properties establish a connection between the Adapter Connection wizard, a tool that is used to create business objects, and the SAP server. The properties that you configure in the Adapter Connection wizard specify such things as connection configuration, and tracing and logging options.

After you have established a connection between the Adapter Connection wizard and the SAP server, the wizard can access the metadata that it needs from the SAP server to create business objects.

Some of the properties that you set in the Adapter Connection wizard are used as the initial values for resource adapter, managed connection factory, and activation specification properties that you can specify at a later time in the wizard.

The connection properties and their purpose are described in the following table. A complete description of each property is provided in the sections that follow the table. If you set any of these connection properties using bidirectional script, you must set values that identify the format of the bidirectional script that is entered for that property.

Table 37. Connection properties for the Adapter for SAP Software

Property name	Description
"Bidi direction " on page 1336	The orientation component of the bidi format specification.
"Bidi ordering schema" on page 1336	The ordering schema of the bidi format specification.
"Bidi numeric shaping" on page 1337	The numeric shaping component of the bidi format specification.
"Bidi shaping" on page 1337	The shaping component of the bidi format specification.
"Bidi symmetric swapping" on page 1337	The symmetric swapping component of the bidi format specification.
"Client" on page 1338	The client number of the SAP system to which the adapter connects.
"Codepage number" on page 1338	Indicates the numeric identifier of the code page.
"Folder for RFC trace files" on page 1339	Sets the fully qualified local path to the folder into which the RFC trace files are written.
"Host name" on page 1339	Specifies the IP address or the name of the application server host that the adapter logs on to.
"Language code" on page 1339	Specifies the language in which the adapter logs on.
"Log file output location property" on page 1340	The location of the log file for enterprise metadata discovery.
"Logging level property" on page 1340	The type error for which logging occurs during enterprise metadata discovery.
"Password" on page 1341	The password of the user account of the adapter on the SAP application server.
"RFC trace level" on page 1341	Specifies the global trace level.
"RFC trace on" on page 1342	Specifies whether to generate a text file detailing the RFC activity for each event listener.
"SAP interface name" on page 1342	The SAP interface to be used.
"System number" on page 1343	The system number of the SAP application server.
"User name" on page 1343	The user account for the adapter on the SAP server.

The Adapter Connection wizard uses the bidirectional connection properties to apply the correct bidirectional transformation on the data that is passed to the SAP server.

The bidi properties specify the bidirectional format for data coming from an external application into the adapter in the form of any business object that is supported by this adapter.

Accept the default values for the bidirectional formatting properties on the Adapter Connection wizard that provides SAP server bidirectional format specification. When combined, these bidirectional properties define one single bidirectional format.

The default values for bidirectional formatting properties listed in this section are based on Windows bidirectional formatting. If the Enterprise Information System supports a bidirectional format other than the Windows standard bidirectional format, you must make appropriate changes to the bidi properties that are listed in the following sections.

Bidi direction

This property specifies the orientation component of the bidi format specification.

Table 38. Bidi direction details

Required	No
Possible values	Possible values include: • `LTR` The orientation is left-to-right. • `RTL` The orientation is right-to-left. • `contextualLTR` The orientation is left-to-right because of the context. A character that is not categorized as LTR, and hat is located between two strong characters with a different writing direction, inherits the main context's writing direction (in a LTR document the character becomes LTR). • `contextualRTL` The orientation is right-to-left because of the context. A character that is not categorized as RTL, and is located between two strong characters with a different writing direction, inherits the main context's writing direction (in a RTL document the character becomes RTL).
Default	`LTR`
Property type	String
Usage	Specifies the orientation component of the bidi format specification.
Globalized	Yes
Bidi supported	No

Bidi ordering schema

This property specifies the ordering schema of the bidi format specification.

Table 39. Bidi ordering schema details

Required	No
Possible values	`Implicit` `Visual`

Table 39. Bidi ordering schema details (continued)

Default	`Implicit`
Property type	String
Usage	Specifies the ordering schema of the bidi format specification.
Globalized	Yes
Bidi supported	No

Bidi numeric shaping

This property specifies the numeric shaping component of the bidi format specification.

Table 40. Bidi numeric details

Required	No

Possible values	`Nominal`
	`National`
	`Contextual`

Default	`Nominal`
Property type	String
Usage	Specifies the numeric shaping component of the bidi format specification.
Globalized	Yes
Bidi supported	No

Bidi shaping

This property specifies the shaping component of the bidi format specification.

Table 41. Bidi shaping details

Required	No

Possible values	`Nominal`
	`Shaped`
	`Initial`
	`Middle`
	`Final`
	`Isolated`

Default	`Nominal`
Property type	String
Usage	Specifies the shaping component of the bidi format specification.
Globalized	Yes
Bidi supported	No

Bidi symmetric swapping

This property specifies the symmetric swapping component of the bidi format specification.

Table 42. Bidi symmetric swapping details

Required	No

Table 42. Bidi symmetric swapping details (continued)

Possible values	`True` `False`
Default	`True`
Property type	Boolean
Usage	This property specifies the symmetric swapping component of the bidi format specification.
Globalized	Yes
Bidi supported	No

Client

This property is the client number of the SAP system to which the adapter connects.

Table 43. Client details

Required	Yes
Possible values	You can enter a range of values from `000` to `999`.
Default	`100`
Property type	Integer
Usage	When an application attempts to log on to the SAP server, the application must have a Client number associated with it. The Client property value identifies the client (the adapter) that is attempting to log onto the SAP server.
Globalized	No
Bidi supported	No

Codepage number

The numeric identifier of the code page.

Table 44. Codepage number details

Required	No
Possible values	You can enter a range of values from `0000` to `9999`. For a full listing of languages and associated code page numbers that are supported by SAP, see SAP Note 7360.
Default	The default value for this property is conditionally determined by the value set for the **Language code** property.
Property type	Integer
Usage	The value that is assigned to the **Codepage number** defines the code page to be used and has a one-to-one relationship with the value set for the **Language code** property. The **Codepage number** establishes a connection to the appropriate language. Each language code value has a code page number value associated with it. For example, the language code for English is EN. If you select EN (English) as your language code, the code page number is automatically set to the numeric value that is associated with EN (English). The SAP code page number for EN (English) is `1100`.
Example	If **Language code** is set to `JA` (Japanese), **Codepage number** is set to `8000`.
Globalized	No
Bidi supported	No

Folder for RFC trace files

This property sets the fully qualified local path to the folder in which to write RFC trace files.

Table 45. Folder for RFC trace files details

Required	No
Default	No default value
Property type	String
Usage	Identifies the fully qualified local path into which RFC trace files are written. If **RFC trace on** is set to False (not selected), you are not permitted to set a value in the **Folder for RFC trace files** property.
Example	`c:\temp\rfcTraceDir`
Globalized	Yes
Bidi supported	No

Host name

Specifies the IP address or the name of the application server host that the adapter logs on to.

Table 46. Host name details

Required	Yes (when load balancing is not used).
Default	No default value
Property type	String
Usage	When the adapter is configured to run without load balancing, this property specifies the IP address or the name of the application server that the adapter logs on to.
Example	`sapServer`
Globalized	No
Bidi supported	No

Language code

SAP logon language code.

Table 47. Language code details

Required	Yes
Possible values	Each of the supported languages is preceded by a 2 character language code. The language itself is displayed in parentheses. The language codes in the list represent the SAP default set of 41 languages for non Unicode systems plus Arabic. For a full listing of supported language codes and languages, see the SAP documentation.
Default	The default language code will be your current locale. If your current locale is not listed as one of the supported language codes, then a default language code of `EN (English)` is used.
Property type	String
Usage	If you manually enter a language code, you do not need to enter the language in parentheses.
Example	If the system locale is English, the value for this property is `EN (English)`

Table 47. Language code details (continued)

Globalized	No
Bidi supported	No

Log file output location property

This property specifies the location of the log file for external metadata discovery.

Table 48. Log file output location details

Required	Yes
Default	The .metadata directory of the workspace
Property type	String
Usage	Use this directory to hold the log file that lists the errors that occur during the discovery process. The type of discovery errors for which logging occurs is controlled by the **Logging level** property
Example	`C:\IBM\wmbt61\workspace\.metadata\SAPMetadataDiscovery.log`
Globalized	Yes
Bidi supported	No

Logging level property

This property specifies the type error for which logging occurs during enterprise metadata discovery.

Table 49. Logging level details

Required	No

Possible values	`FATAL`
	`SEVERE`
	`WARNING`
	`AUDIT`
	`INFO`
	`CONFIG`
	`DETAIL`

Default	`SEVERE`
Property type	String
Usage	Use this property to tailor tracing capabilities. When you specify an error type, you indicate that trace operations occur only for errors of the specified type.

Table 49. Logging level details (continued)

Example	If you accept the default value of SEVERE, trace information is provided on errors that fall into the SEVERE category. Severe errors mean that an operation cannot continue, although the adapter can still function. Severe errors also include error conditions that indicate an impending fatal error, that is, reporting on situations that strongly suggest that resources are on the verge of being depleted. Other error descriptions are listed here: • Fatal Adapter cannot continue. Adapter cannot function • Warning Potential error or impending error, including conditions that indicate a progressive failure (for example, the potential leaking of resources). • Audit Significant event affecting adapter state or resources • Info General information outlining overall operation progress. • Config Configuration change or status. • Detail General information detailing operation progress
Globalized	Yes
Bidi supported	No

Password

This property is the password of the user account of the adapter on the SAP application server.

Table 50. Password details

Required	Yes
Default	No default value
Property type	String
Usage	The restrictions on the password depend on the version of SAP Web Application Server. • For SAP Web Application Server version 6.40 or earlier, the password: – Must be uppercase – Must be 8 characters in length • For versions of SAP Web Application Server later than 6.40, the password: – Is not case-sensitive – Can be up to 40 characters in length
Globalized	No
Bidi supported	Yes

RFC trace level

This property specifies the global trace level.

Table 51. RFC trace level details

Required	No

Table 51. RFC trace level details (continued)

Possible values	`1` - This is the default RFC trace level. When specified, SAP JCo Java API logging occurs. `3` - When specified, SAP JCo JNI API logging occurs. `5` - When specified, error diagnostic logging occurs.
Default	1
Property type	Integer
Usage	If **RFC trace on** is set to `False` (not selected), you cannot set a value in the **RFC trace level** property.
Globalized	No
Bidi supported	No

RFC trace on

This property specifies whether to generate a text file detailing the RFC activity for each event listener.

Table 52. RFC trace on details

Required	No
Possible values	`True` `False`
Default	`False`
Property type	Boolean
Usage	A value of `True` activates tracing, which generates a text file. This file is created in the directory in which the adapter process was started. The file has a prefix of `rfx` and a file type of trc (for example, rfc03912_02220.trc). Use these text files in a development environment only, because the files can grow rapidly. If **RFC trace on** is set to `False` (not selected), you cannot set values in the **Folder for RFC trace files** or **RFC trace level** properties.
Example	Examples of the information in the file are RfcCall FUNCTION BAPI_CUSTOMER_GETLIST, followed by the information for the parameters in the interface, or RFC Info rfctable, followed by the data from one of the interface tables. The trace file is created in the directory where the adapter process has been started. The trace file has a .trc file extension and the file name will start with the letters `rfc` followed by a unique identifier. For example, `rfc03912_02220.trc`.
Globalized	No
Bidi supported	No

SAP interface name

This property indicates whether you are creating business objects for the ALE, BAPI, Advanced event processing, or Query interface for SAP Software.

Table 53. SAP interface name details

Required	Yes

Table 53. SAP interface name details (continued)

Possible values	For outbound:
	`Advanced event processing (AEP)`
	`ALE`
	`ALE pass-through IDoc`
	`BAPI`
	`BAPI work unit`
	`BAPI result set`
	`Query interface for SAP Software (QSS)`
	For inbound:
	`Advanced event processing (AEP)`
	`ALE`
	`ALE pass-through IDoc`

Default	For outbound: `BAPI` For inbound:`ALE`
Property type	String
Usage	Specifies the interface used by the adapter. The adapter interacts with the interface to support outbound or inbound processing by enabling the exchange of data in the form of business objects.
Globalized	No
Bidi supported	No

System number

This property is the system number of the SAP application server.

Table 54. System number details

Required	Yes
Possible values	You can enter a range of values from `00` to `99`.
Default	`00`
Property type	Integer
Usage	The system number further identifies the Gateway service.
Globalized	No
Bidi supported	No

User name

This property is the user account for the adapter on the SAP server.

Table 55. User name details

Required	Yes
Default	No default value
Property type	String

Table 55. User name details (continued)

Usage	Maximum length of 12 characters. The user name is not case sensitive. It is recommended that you set up a CPIC user account in the SAP application and that you give this account the necessary privileges to manipulate the data required by the business objects supported by the adapter. For example, if the adapter must perform certain SAP business transactions, the adapter's account in the SAP application must have the permissions set to allow it to perform these transactions.
Example	`SapUser`
Globalized	Yes
Bidi supported	Yes

Resource adapter properties (SAP):

The resource adapter properties control the general operation of the adapter. Use the Adapter Connection wizard to set the resource adapter properties when you configure the adapter.

Adapter ID to use for logging and tracing (AdapterID)

This property identifies a specific deployment, or instance, of the adapter.

Table 56. Adapter ID to use for logging and tracing details

Required	Yes
Default	CWYMY_Adapter Without local transaction support: CWYAP_SAPAdapter With local transaction support: CWYAP_SAPAdapter_Tx
Property type	String
Usage	Use this property to identify the adapter instance for PMI events. If you are deploying multiple instances of an adapter, set this property to a unique value for each adapter instance. For inbound processing, this property is retrieved from the resource adapter properties. For outbound processing, it is retrieved from the managed connection factory properties.
Globalized	Yes
Bidi supported	No

Managed connection factory properties (SAP):

The adapter uses the managed connection factory properties at run time to create an outbound connection instance with the SAP server.

Use the Adapter Connection wizard to set the managed connection factory properties.

The following table lists and describes the managed connection factory properties. A more detailed description of each property is provided in the sections that follow the table.

Table 57. Managed connection factory properties forAdapter for SAP Software

Property name	Description
ABAPDebug	ABAP debugger property.

Table 57. Managed connection factory properties forAdapter for SAP Software (continued)

Property name	Description
Client	The client number of the SAP system to which the adapter connects.
Codepage	Indicates the numeric identifier of the code page.
SncMode	Indicates whether secure network connection mode is used.
RfcTracePath	Sets the fully qualified local path to the folder into which the RFC trace files are written.
GatewayHost	The host name of the SAP gateway.
GatewayService	The identifier of the gateway on the gateway host that carries out the RFC services.
ApplicationServerHost	Specifies the IP address or the name of the application server host that the adapter logs on to.
Language code	Specifies the Language code in which the adapter logs on to SAP.
MessageServerHost	Specifies the name of the host on which the message server is running.
PartnerCharset	Specifies PartnerCharset encoding.
Password	The password of the user account of the adapter on the SAP application server.
RfcTraceLevel	Specifies the global trace level.
RfcTraceOn	Specifies whether to generate a text file detailing the RFC activity for each event listener.
SAPSystemID	Specifies the system ID of the SAP system for which logon load balancing is allowed.
SncLib	Specifies the path to the library that provides the secure network connection service.
SncMyname	Specifies the name of the secure network connection.
SncPartnername	Specifies the name of the secure network connection partner.
SncQop	Specifies the level of security for the secure network connection.
SystemNumber	The system number of the SAP application server.
Use wait parameter before calling BAPI commit	Specifies whether the WAIT parameter is set on a BAPI_TRANSACTION_COMMIT.
userName	The user account for the adapter on the SAP server.
X509cert	Specifies the X509 certificate to be used as the logon ticket.

ABAP debug

This property specifies whether the adapter invokes the ABAP Debugger for the appropriate function module when the adapter starts to process a business object.

Table 58. ABAP debug details

Required	No

Table 58. ABAP debug details (continued)

Possible values	`True` `False`
Default	`False`
Property type	Boolean
Usage	When the property is set to `True`, the adapter opens the SAP GUI in debug mode. You must have appropriate authorization to use the debugger. Create a dialog user ID because a CPI-C user ID cannot open an SAP GUI session. You must have authorization to run in debug mode as well as any authorizations that are required by the ABAP code that is being debugged. For example, if a BAPI_CUSTOMER_CREATEFROMDATA1 is being debugged, you must have authorization to create customers. You can add breakpoints only after the debugger opens. Always set this property to `False` in a production environment. This property is supported on Windows systems only.
Globalized	No
Bidi supported	No

Client

This property is the client number of the SAP system to which the adapter connects.

Table 59. Client details

Required	Yes
Possible values	You can enter a range of values from `000` to `999`.
Default	`100`
Property type	Integer
Usage	When an application attempts to log on to the SAP server, the application must have a Client number associated with it. The Client property value identifies the client (the adapter) that is attempting to log onto the SAP server.
Globalized	No
Bidi supported	No

Codepage number

The numeric identifier of the code page.

Table 60. Codepage number details

Required	No
Possible values	You can enter a range of values from `0000` to `9999`. For a full listing of languages and associated code page numbers that are supported by SAP, see SAP Note 7360.
Default	The default value for this property is conditionally determined by the value set for the **Language code** property.
Property type	Integer

Table 60. Codepage number details (continued)

Usage	The value that is assigned to the **Codepage number** defines the code page to be used and has a one-to-one relationship with the value set for the **Language code** property. The **Codepage number** establishes a connection to the appropriate language. Each language code value has a code page number value associated with it. For example, the language code for English is EN. If you select EN (English) as your language code, the code page number is automatically set to the numeric value that is associated with EN (English). The SAP code page number for EN (English) is `1100`.
Example	If **Language code** is set to `JA` (Japanese), **Codepage number** is set to `8000`.
Globalized	No
Bidi supported	No

Enable Secure Network Connection

This property indicates whether secure network connection mode is enabled.

Table 61. Enable Secure Network Connection details

Required	No
Possible values	`0` (off) `1` (on)
Default	`0`
Property type	String
Usage	Set the value to 1 (on) if you want to use secure network connection. If you set this value to 1, you must also set following properties: • "Secure Network Connection library path" on page 1351 • "Secure Network Connection name" on page 1352 • "Secure Network Connection partner" on page 1352 • "Secure Network Connection security level" on page 1352
Globalized	No
Bidi supported	No

Folder for RFC trace files

This property sets the fully qualified local path to the folder in which to write RFC trace files.

Table 62. Folder for RFC trace files details

Required	No
Default	No default value
Property type	String
Usage	Identifies the fully qualified local path into which RFC trace files are written. If **RFC trace on** is set to False (not selected), you are not permitted to set a value in the **Folder for RFC trace files** property.
Example	`c:\temp\rfcTraceDir`
Globalized	Yes
Bidi supported	No

Gateway host

This property is the Gateway host name. Enter either the IP address or the name of the Gateway host. Consult your SAP administrator for information on the Gateway host name.

Table 63. Gateway host details

Required	Yes
Default	No default value
Property type	String
Usage	This property is the host name of the SAP gateway. The gateway enables communication between work processes on the SAP system and external programs. The host identified is used as the gateway for the resource adapter. Maximum length of 20 characters. If the computer name is longer than 20 characters, define a symbolic name in the THOSTS table.
Globalized	No
Bidi supported	No

Gateway service

This property is the identifier of the gateway on the gateway host that carries out the RFC services.

Table 64. Gateway service details

Required	Yes
Default	`sapgw00`
Property type	String
Usage	These services enable communication between work processes on the SAP server and external programs. The service typically has the format of `sapgw00`, where 00 is the SAP system number. Maximum of 20 characters.
Globalized	No
Bidi supported	No

Host name

Specifies the IP address or the name of the application server host that the adapter logs on to.

Table 65. Host name details

Required	Yes (when load balancing is not used).
Default	No default value
Property type	String
Usage	When the adapter is configured to run without load balancing, this property specifies the IP address or the name of the application server that the adapter logs on to.
Example	`sapServer`
Globalized	No
Bidi supported	No

Language code

This property specifies the Language code in which the adapter logs on.

Table 66. Language code details

Required	Yes
Possible values	For a full listing of languages and associated code page numbers that are supported by SAP, see SAP Note 7360.
Default	The default value for the **Language code** property is based on the system locale.
Property type	String
Usage	Each of the supported languages is preceded by a two-character language code. The language itself is displayed in parentheses. If you enter a language code manually, you do not need to enter the language in parentheses. The language codes that are listed represent the SAP default set of 41 languages for non-Unicode systems plus Arabic. The value that you select determines the value of the **Codepage number** property.
Example	If the system locale is English, the value for this property is `EN (English)`.
Globalized	No
Bidi supported	No

Message server host

This property specifies the name of the host on which the message server is running.

Table 67. Message server host details

Required	Yes (if load balancing is used)
Default	No default value
Property type	String
Usage	This property specifies the name of the host that will inform all the servers (instances) belonging to this SAP system of the existence of the other servers to be used for load balancing. The message server host contains the information about load balancing for RFC clients so that an RFC client can be directed to an appropriate application server.
Example	`SAPERP05`
Globalized	No
Bidi supported	No

Partner character set

This property specifies the partner character set encoding.

Table 68. Partner character set details

Required	No
Default	UTF-8
Property type	String
Usage	When an encoding is specified, it is used; otherwise, the default encoding is used.

Table 68. Partner character set details (continued)

Globalized	No
Bidi supported	No

Password

This property is the password of the user account of the adapter on the SAP application server.

Table 69. Password details

Required	Yes
Default	No default value
Property type	String
Usage	The restrictions on the password depend on the version of SAP Web Application Server. • For SAP Web Application Server version 6.40 or earlier, the password: – Must be uppercase – Must be 8 characters in length • For versions of SAP Web Application Server later than 6.40, the password: – Is not case-sensitive – Can be up to 40 characters in length
Globalized	No
Bidi supported	Yes

RFC trace level

This property specifies the global trace level.

Table 70. RFC trace level details

Required	No
Possible values	`1` - This is the default RFC trace level. When specified, SAP JCo Java API logging occurs. `3` - When specified, SAP JCo JNI API logging occurs. `5` - When specified, error diagnostic logging occurs.
Default	1
Property type	Integer
Usage	If **RFC trace on** is set to `False` (not selected), you cannot set a value in the **RFC trace level** property.
Globalized	No
Bidi supported	No

RFC trace on

This property specifies whether to generate a text file detailing the RFC activity for each event listener.

Table 71. RFC trace on details

Required	No
Possible values	`True` `False`

Table 71. RFC trace on details (continued)

Default	`False`
Property type	Boolean
Usage	A value of `True` activates tracing, which generates a text file. This file is created in the directory in which the adapter process was started. The file has a prefix of `rfx` and a file type of trc (for example, rfc03912_02220.trc). Use these text files in a development environment only, because the files can grow rapidly. If **RFC trace on** is set to `False` (not selected), you cannot set values in the **Folder for RFC trace files** or **RFC trace level** properties.
Example	Examples of the information in the file are RfcCall FUNCTION BAPI_CUSTOMER_GETLIST, followed by the information for the parameters in the interface, or RFC Info rfctable, followed by the data from one of the interface tables. The trace file is created in the directory where the adapter process has been started. The trace file has a .trc file extension and the file name will start with the letters `rfc` followed by a unique identifier. For example, `rfc03912_02220.trc`.
Globalized	No
Bidi supported	No

SAP system ID

This property specifies the system ID of the SAP system for which logon load balancing is allowed.

Table 72. SAP system ID details

Required	Yes (when load balancing is used)
Default	No default value
Property type	String
Usage	Value must be three characters
Example	`DYL`
Globalized	No
Bidi supported	No

Secure Network Connection library path

This property specifies the path to the library that provides the secure network connection service.

Table 73. Secure Network Connection library path details

Required	Yes, if SncMode is set to 1; no otherwise.
Default	No default value
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), specify the path to the library that provides the service.
Example	`/WINDOWS/system32/gssapi32.dll`
Globalized	No
Bidi supported	No

Secure Network Connection name

This property specifies the name of the secure network connection.

Table 74. Secure Network Connection name details

Required	Yes, if SncMode is set to 1; no otherwise.
Default	No default value
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), specify a name for the connection.
Example	`DOMAINNAME/USERNAME`
Globalized	No
Bidi supported	No

Secure Network Connection partner

This property specifies the name of the secure network connection partner.

Table 75. Secure Network Connection partner details

Required	Yes, if SncMode is set to 1; no otherwise.
Default	No default value
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), specify a name for the connection partner.
Example	`CN=sap00.saperpdev, OU=Adapter, O=IBM, C=US`
Globalized	No
Bidi supported	No

Secure Network Connection security level

This property specifies the level of security for the secure network connection.

Table 76. Secure Network Connection security level details

Required	Yes, if SncMode is set to 1; no otherwise.
Possible values	`1` (Authentication only) `2` (Integrity protection) `3` (Privacy protection) `8` (Use the value from `snc/data_protection/use` on the application server) `9` (Use the value from `snc/data_protection/max` on the application server)
Default	`3` (Privacy protection)
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), specify a value to indicate the level of security for the connection.
Globalized	No
Bidi supported	No

System number

This property is the system number of the SAP application server.

Table 77. System number details

Required	Yes
Possible values	You can enter a range of values from `00` to `99`.
Default	`00`
Property type	Integer
Usage	The system number further identifies the Gateway service.
Globalized	No
Bidi supported	No

Use wait parameter before calling BAPI commit

This property indicates whether the adapter calls a BAPI_TRANSACTION_COMMIT with the WAIT parameter set.

Table 78. Use wait parameter before calling BAPI commit

Required	No
Possible values	`True` `False`
Default	`False`
Property type	Boolean
Usage	This property applies to BAPI outbound synchronous RFC processing only. When you set this property to `True`, the Adapter for SAP Software calls a BAPI_TRANSACTION_COMMIT with the WAIT parameter set, so that other resource managers are committed only when the update has been completed in SAP. If you accept the default value of `False`, message flow processing might continue before the update has been completed in SAP. Therefore, after calling a BAPI, the updated data might not be available when you next access the system.
Globalized	No
Bidi supported	No

User name

This property is the user account for the adapter on the SAP server.

Table 79. User name details

Required	Yes
Default	No default value
Property type	String
Usage	Maximum length of 12 characters. The user name is not case sensitive. It is recommended that you set up a CPIC user account in the SAP application and that you give this account the necessary privileges to manipulate the data required by the business objects supported by the adapter. For example, if the adapter must perform certain SAP business transactions, the adapter's account in the SAP application must have the permissions set to allow it to perform these transactions.

Table 79. User name details (continued)

Example	`SapUser`
Globalized	Yes
Bidi supported	Yes

X509 certificate

This property specifies the X509 certificate to be used as the logon ticket.

Table 80. X509 certificate details

Required	No.
Default	No default value
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), you can provide a value for the X509 certificate.
Globalized	No
Bidi supported	No

Activation specification properties for BAPI inbound processing:

Activation specification properties hold the inbound event processing configuration information for a message endpoint.

Activation specification properties are used during endpoint activation to notify the adapter of eligible event listeners. During inbound processing, the adapter uses these event listeners to receive events before forwarding them to the endpoint.

You set the activation specification properties using the Adapter Connection wizard.

Table 81 lists and describes the activation specification properties that apply to both synchronous RFC and asynchronous transactional RFC. Table 82 on page 1355 applies only to asynchronous transaction RFC properties that are used for assured-once delivery.

A more detailed description of each property is provided in the sections that follow the tables.

Table 81. Activation specification properties for BAPI inbound processing

Property name	Description
"Client" on page 1357	The client number of the SAP system to which the adapter connects.
"Codepage number" on page 1357	Indicates the numeric identifier of the code page.
"Enable Security Network Connection" on page 1358	Indicates whether secure network connection mode is used.
"Folder for RFC trace files" on page 1360	Sets the fully qualified local path to the folder into which the RFC trace files are written.
"Gateway host" on page 1360	The host name of the SAP gateway.
"Gateway service" on page 1360	The identifier of the gateway on the gateway host that carries out the RFC services.

Table 81. Activation specification properties for BAPI inbound processing (continued)

Property name	Description
"Host name" on page 1361	Specifies the IP address or the name of the application server host that the adapter logs on to.
"Language code" on page 1361	Specifies the Language code in which the adapter logs on to SAP.
"Logon group name" on page 1361	An identifier of the name of the group of application server instances that have been defined in transaction SMLG and linked together for logon load balancing.
"Maximum number of retries in case of system connection failure" on page 1362	Specifies the number of times the adapter tries to restart the event listeners.
"Message server host" on page 1362	Specifies the name of the host on which the message server is running.
"Number of listeners" on page 1363	Specifies the number of event listeners that are to be started.
"Partner character set" on page 1363	Specifies PartnerCharset encoding.
"Password" on page 1363	The password of the user account of the adapter on the SAP application server.
"RFC program ID" on page 1364	The remote function call identifier under which the adapter registers in the SAP gateway.
"RFC trace level" on page 1365	Specifies the global trace level.
"RFC trace on" on page 1365	Specifies whether to generate a text file detailing the RFC activity for each event listener.
"SAP system ID" on page 1365	Specifies the system ID of the SAP system for which logon load balancing is allowed.
"Secure Network Connection library path" on page 1366	Specifies the path to the library that provides the secure network connection service.
"Secure Network Connection name" on page 1366	Specifies the name of the secure network connection.
"Secure Network Connection partner" on page 1366	Specifies the name of the secure network connection partner.
"Secure Network Connection security level" on page 1367	Specifies the level of security for the secure network connection.
"System number" on page 1367	The system number of the SAP application server.
"Time between retries in case of system connection failure (milliseconds)" on page 1367	Specifies the time interval between attempts to restart the event listeners.
"User name" on page 1368	The user account for the adapter on the SAP server.
"X509 certificate" on page 1369	Specifies the X509 certificate to be used as the logon ticket.

The properties in the following table applies only to assured-once delivery. When you select assured-once delivery, the transaction ID sent from the SAP server is stored in a data source. You specify information about the data source with these properties.

Table 82. Additional activation specification properties for assured-once delivery

Property name	Description
"Assured once-only delivery (AssuredOnceDelivery)" on page 1356	Specifies whether to provide assured once-only delivery for inbound events.

Table 82. Additional activation specification properties for assured-once delivery (continued)

Property name	Description
"Auto create event table"	Indicates whether the adapter should create the event recovery table automatically if it does not already exist.
"Event recovery data source (JNDI) name" on page 1359	The schema used for automatically creating the event recovery table.
"Event recovery data source (JNDI) name" on page 1359	The JNDI name of the data source configured for event recovery.
"Event recovery table name" on page 1359	The name of the event recovery table.
"Password used to connect to event data source" on page 1364	The user password for connecting to the database.
"User name used to connect to event data source" on page 1368	The user name for connecting to the database.

Assured once-only delivery (AssuredOnceDelivery)

This property specifies whether to provide assured once-only delivery for inbound events.

Table 83. Assured once-only delivery details

Required	No
Default	`False`
Property type	Boolean
Usage	When this property is set to `True`, the adapter provides assured once-only event delivery, so that each event is delivered only once. A value of `False` does not provide assured once-only event delivery, but provides better performance. When this property is set to `True`, the adapter attempts to store transaction (XID) information in the event store. If it is set to `False`, the adapter does not attempt to store the information. This property is used only if the export component is transactional. If the export component is not transactional, no transaction can be used, regardless of the value of this property.
Globalized	No
Bidi supported	No

Note: The **Assured once-only delivery** property applies only to asynchronous transactional RFC processing.

Auto create event table

Determines if the event table is created automatically.

Table 84. Auto create event table details

Required	Yes, if **Assured once-only event delivery** is set to `True`, No otherwise.
Possible values	`True` `False`
Default	`True`
Property type	Boolean

Table 84. Auto create event table details (continued)

Usage	This property indicates whether the adapter should create the event recovery table automatically if it does not already exist. In the administrative console, this property is listed as "EP_CreateTable". If you specify a value of `True` to automatically create the table, you must specify information about the event table (such as the event recovery table name). The value provided in the Event recovery table name property is used to create the table.
Globalized	No
Bidi supported	No

Note: The **Auto create event table** property applies only to asynchronous transactional RFC processing.

Client

This property is the client number of the SAP system to which the adapter connects.

Table 85. Client details

Required	Yes
Possible values	You can enter a range of values from `000` to `999`.
Default	`100`
Property type	Integer
Usage	When an application attempts to log on to the SAP server, the application must have a Client number associated with it. The Client property value identifies the client (the adapter) that is attempting to log onto the SAP server.
Globalized	No
Bidi supported	No

Codepage number

The numeric identifier of the code page.

Table 86. Codepage number details

Required	No
Possible values	You can enter a range of values from `0000` to `9999`. For a full listing of languages and associated code page numbers that are supported by SAP, see SAP Note 7360.
Default	The default value for this property is conditionally determined by the value set for the **Language code** property.
Property type	Integer

Table 86. Codepage number details (continued)

Usage	The value that is assigned to the **Codepage number** defines the code page to be used and has a one-to-one relationship with the value set for the **Language code** property. The **Codepage number** establishes a connection to the appropriate language. Each language code value has a code page number value associated with it. For example, the language code for English is EN. If you select EN (English) as your language code, the code page number is automatically set to the numeric value that is associated with EN (English). The SAP code page number for EN (English) is `1100`.
Example	If **Language code** is set to `JA` (Japanese), **Codepage number** is set to `8000`.
Globalized	No
Bidi supported	No

Database schema name

This property is the schema used for automatically creating the event recovery table.

Note: In the administrative console, this property is listed as "EP_SchemaName".

Table 87. Database schema name details

Required	No
Default	No default value.
Property type	String
Usage	Specifies the schema name for the database used by the adapters event persistence feature.
Example	`ALE_SCHEMA`
Globalized	Yes
Bidi supported	No

Note: The **Database schema name** property applies only to asynchronous transactional RFC processing.

Enable Security Network Connection

This property indicates whether secure network connection mode is enabled.

Table 88. Enable Security Network Connection details

Required	No
Possible values	`0` (off) `1` (on)
Default	`0`
Property type	String
Usage	Set the value to 1 (on) if you want to use secure network connection. If you set this value to 1, you must also set following properties: • "Secure Network Connection library path" on page 1366 • "Secure Network Connection name" on page 1366 • "Secure Network Connection partner" on page 1366 • "Secure Network Connection security level" on page 1367

Table 88. Enable Security Network Connection details (continued)

Globalized	No
Bidi supported	No

Event recovery data source (JNDI) name

This property is the JNDI name of the data source configured for event recovery.

Note: In the administrative console, this property is listed as "EP_DataSource_JNDIName".

Table 89. Event recovery data source (JNDI) name details

Required	Yes
Default	No default value.
Property type	String
Usage	Used in event recovery processing. The data source must be created in WebSphere Process Server. The adapter utilizes data source for *persisting* the event state.
Example	`jdbc/DB2`
Globalized	No
Bidi supported	No

Note: The **Event recovery data source (JNDI) name** property applies only to asynchronous transactional RFC processing.

Event recovery table name

This property is the name of the event recovery table.

Note: In the administrative console, this property is listed as "EP_TableName".

Table 90. Event recovery table name details

Required	Yes
Default	No default value.
Property type	String
Usage	Used in event recovery processing. Consult database documentation for information on naming conventions. It is recommended that a separate event recovery table is configured for each endpoint. The same data source can be used to hold all of the event recovery tables.
Example	`EVENT_TABLE`
Globalized	No
Bidi supported	No

Note: The **Event recovery table name** property applies only to asynchronous transactional RFC processing.

Folder for RFC trace files

This property sets the fully qualified local path to the folder in which to write RFC trace files.

Table 91. Folder for RFC trace files details

Required	No
Default	No default value
Property type	String
Usage	Identifies the fully qualified local path into which RFC trace files are written. If **RFC trace on** is set to False (not selected), you are not permitted to set a value in the **Folder for RFC trace files** property.
Example	`c:\temp\rfcTraceDir`
Globalized	Yes
Bidi supported	No

Gateway host

This property is the Gateway host name. Enter either the IP address or the name of the Gateway host. Consult your SAP administrator for information on the Gateway host name.

Table 92. Gateway host details

Required	Yes
Default	No default value
Property type	String
Usage	This property is the host name of the SAP gateway. The gateway enables communication between work processes on the SAP system and external programs. The host identified is used as the gateway for the resource adapter. Maximum length of 20 characters. If the computer name is longer than 20 characters, define a symbolic name in the THOSTS table.
Globalized	No
Bidi supported	No

Gateway service

This property is the identifier of the gateway on the gateway host that carries out the RFC services.

Table 93. Gateway service details

Required	Yes
Default	`sapgw00`
Property type	String
Usage	These services enable communication between work processes on the SAP server and external programs. The service typically has the format of `sapgw00`, where 00 is the SAP system number. Maximum of 20 characters.
Globalized	No

Table 93. Gateway service details (continued)

Bidi supported	No

Host name

Specifies the IP address or the name of the application server host that the adapter logs on to.

Table 94. Host name details

Required	Yes (when load balancing is not used).
Default	No default value
Property type	String
Usage	When the adapter is configured to run without load balancing, this property specifies the IP address or the name of the application server that the adapter logs on to.
Example	`sapServer`
Globalized	No
Bidi supported	No

Language code

This property specifies the Language code in which the adapter logs on.

Table 95. Language code details

Required	Yes
Possible values	For a full listing of languages and associated code page numbers that are supported by SAP, see SAP Note 7360.
Default	The default value for the **Language code** property is based on the system locale.
Property type	String
Usage	Each of the supported languages is preceded by a two-character language code. The language itself is displayed in parentheses. If you enter a language code manually, you do not need to enter the language in parentheses. The language codes that are listed represent the SAP default set of 41 languages for non-Unicode systems plus Arabic. The value that you select determines the value of the **Codepage number** property.
Example	If the system locale is English, the value for this property is `EN (English)`.
Globalized	No
Bidi supported	No

Logon group name

This property is an identifier for the name of the group of application server instances that have been defined in transaction SMLG and linked together for logon load balancing.

Table 96. Logon group details

Required	Yes (if load balancing is used)

Table 96. Logon group details (continued)

Possible values	Consult SAP documentation for information on creating Logon groups and on calling transaction SMLG.
Default	No default value
Property type	String
Usage	When the adapter is configured for load balancing, this property represents the name of the group of application server instances that have been defined in transaction SMLG and linked together for logon load balancing. Logon load balancing allows for the dynamic distribution of logon connections to application server instances. Maximum of 20 characters. On most SAP systems, the SPACE logon group is reserved by SAP.
Globalized	No
Bidi supported	No

Maximum number of retries in case of system connection failure

This property specifies the number of times the adapter tries to restart the event listeners.

Table 97. Maximum number of retries in case of system failure details

Required	Yes
Default	`0`
Property type	Integer
Usage	When the adapter encounters an error related to the inbound connection (if the SAP application is down for example), this property specifies the number of times the adapter tries to restart the event listeners. A value of 0 indicates an infinite number of retries. **Note:** Configure the **Time between retries in case of system connection failure (milliseconds)** appropriately when retrying infinitely. For each retry attempt, the adapter waits based on the time interval specified in the **Time between retries in case of system connection failure (milliseconds)**. **Note:** If all the retry attempts fail, the adapter logs relevant messages and CEI events and stops attempting to recover the event listener. If you reach this point, you might have to restart the application manually.
Globalized	No
Bidi supported	No

Message server host

This property specifies the name of the host on which the message server is running.

Table 98. Message server host details

Required	Yes (if load balancing is used)
Default	No default value
Property type	String

Table 98. Message server host details (continued)

Usage	This property specifies the name of the host that will inform all the servers (instances) belonging to this SAP system of the existence of the other servers to be used for load balancing. The message server host contains the information about load balancing for RFC clients so that an RFC client can be directed to an appropriate application server.
Example	SAPERP05
Globalized	No
Bidi supported	No

Number of listeners

This property specifies the number of listeners that are started by an event.

Table 99. Number of listeners details

Required	No
Default	1
Property type	Integer
Usage	For event sequencing, this property should be set to 1. To improve adapter performance, you can increase the number of listeners.
Globalized	No
Bidi supported	No

Partner character set

This property specifies the partner character set encoding.

Table 100. Partner character set details

Required	No
Default	UTF-8
Property type	String
Usage	When an encoding is specified, it is used; otherwise, the default encoding is used.
Globalized	No
Bidi supported	No

Password

This property is the password of the user account of the adapter on the SAP application server.

Table 101. Password details

Required	Yes
Default	No default value
Property type	String

Table 101. Password details (continued)

Usage	The restrictions on the password depend on the version of SAP Web Application Server. • For SAP Web Application Server version 6.40 or earlier, the password: – Must be uppercase – Must be 8 characters in length • For versions of SAP Web Application Server later than 6.40, the password: – Is not case-sensitive – Can be up to 40 characters in length
Globalized	No
Bidi supported	Yes

Password used to connect to event data source

This property is the user password for connecting to the database.

Note: In the administrative console, this property is listed as "EP_Password".

Table 102. Password to connect to event data source details

Required	Yes
Default	No default value.
Property type	String
Usage	This property specifies the password used by event persistence processing to obtain the database connection from the data source.
Globalized	Yes
Bidi supported	No

Note: The **Password used to connect to event data source** property applies only to asynchronous transactional RFC processing.

RFC program ID

This property is the program identifier under which the adapter registers in the SAP gateway.

Table 103. RFC program ID details

Required	Yes
Possible values	Use the SAP transaction SM59 (Display and Maintain RFC Destinations) to see a list of available RFC program IDs.
Default	No default value.
Property type	String
Usage	The adapter registers with the gateway so that listener threads can process events from RFC-enabled functions. This value must match the program ID registered in the SAP application. The maximum length is 64 characters.
Globalized	No
Bidi supported	No

RFC trace level

This property specifies the global trace level.

Table 104. RFC trace level details

Required	No
Possible values	1 - This is the default RFC trace level. When specified, SAP JCo Java API logging occurs. 3 - When specified, SAP JCo JNI API logging occurs. 5 - When specified, error diagnostic logging occurs.
Default	1
Property type	Integer
Usage	If **RFC trace on** is set to `False` (not selected), you cannot set a value in the **RFC trace level** property.
Globalized	No
Bidi supported	No

RFC trace on

This property specifies whether to generate a text file detailing the RFC activity for each event listener.

Table 105. RFC trace on details

Required	No
Possible values	`True` `False`
Default	`False`
Property type	Boolean
Usage	A value of `True` activates tracing, which generates a text file. This file is created in the directory in which the adapter process was started. The file has a prefix of `rfx` and a file type of trc (for example, rfc03912_02220.trc). Use these text files in a development environment only, because the files can grow rapidly. If **RFC trace on** is set to `False` (not selected), you cannot set values in the **Folder for RFC trace files** or **RFC trace level** properties.
Example	Examples of the information in the file are RfcCall FUNCTION BAPI_CUSTOMER_GETLIST, followed by the information for the parameters in the interface, or RFC Info rfctable, followed by the data from one of the interface tables. The trace file is created in the directory where the adapter process has been started. The trace file has a .trc file extension and the file name will start with the letters `rfc` followed by a unique identifier. For example, `rfc03912_02220.trc`.
Globalized	No
Bidi supported	No

SAP system ID

This property specifies the system ID of the SAP system for which logon load balancing is allowed.

Table 106. SAP system ID details

Required	Yes (when load balancing is used)
Default	No default value
Property type	String
Usage	Value must be three characters
Example	`DYL`
Globalized	No
Bidi supported	No

Secure Network Connection library path

This property specifies the path to the library that provides the secure network connection service.

Table 107. Secure Network Connection library path details

Required	Yes, if SncMode is set to 1; no otherwise.
Default	No default value
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), specify the path to the library that provides the service.
Example	`/WINDOWS/system32/gssapi32.dll`
Globalized	No
Bidi supported	No

Secure Network Connection name

This property specifies the name of the secure network connection.

Table 108. Secure Network Connection name details

Required	Yes, if SncMode is set to 1; no otherwise.
Default	No default value
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), specify a name for the connection.
Example	`DOMAINNAME/USERNAME`
Globalized	No
Bidi supported	No

Secure Network Connection partner

This property specifies the name of the secure network connection partner.

Table 109. Secure Network Connection partner details

Required	Yes, if SncMode is set to 1; no otherwise.
Default	No default value
Property type	String

Table 109. Secure Network Connection partner details (continued)

Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), specify a name for the connection partner.
Example	`CN=sap00.saperpdev, OU=Adapter, O=IBM, C=US`
Globalized	No
Bidi supported	No

Secure Network Connection security level

This property specifies the level of security for the secure network connection.

Table 110. Secure Network Connection security level details

Required	Yes, if SncMode is set to 1; no otherwise.
Possible values	`1` (Authentication only) `2` (Integrity protection) `3` (Privacy protection) `8` (Use the value from `snc/data_protection/use` on the application server) `9` (Use the value from `snc/data_protection/max` on the application server)
Default	`3` (Privacy protection)
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), specify a value to indicate the level of security for the connection.
Globalized	No
Bidi supported	No

System number

This property is the system number of the SAP application server.

Table 111. System number details

Required	Yes
Possible values	You can enter a range of values from `00` to `99`.
Default	`00`
Property type	Integer
Usage	The system number further identifies the Gateway service.
Globalized	No
Bidi supported	No

Time between retries in case of system connection failure (milliseconds)

This property specifies the time interval between attempts to restart the event listeners.

Table 112. Time between retries in case of system connection failure details

Required	Yes
Default	`60000`

Table 112. Time between retries in case of system connection failure details (continued)

Unit of measure	Milliseconds
Property type	Integer
Usage	When the adapter encounters an error related to the inbound connection, this property specifies the time interval the adapter waits in between attempts to restart the event listeners.
Globalized	No
Bidi supported	No

User name

This property is the user account for the adapter on the SAP server.

Table 113. User name details

Required	Yes
Default	No default value
Property type	String
Usage	Maximum length of 12 characters. The user name is not case sensitive. It is recommended that you set up a CPIC user account in the SAP application and that you give this account the necessary privileges to manipulate the data required by the business objects supported by the adapter. For example, if the adapter must perform certain SAP business transactions, the adapter's account in the SAP application must have the permissions set to allow it to perform these transactions.
Example	`SapUser`
Globalized	Yes
Bidi supported	Yes

User name used to connect to event data source

This property is the user name for connecting to the database.

Note: In the administrative console, this property is listed as "EP_UserName".

Table 114. User name to connect to event data source details

Required	Yes
Default	No default value.
Property type	String
Usage	User name used by event persistence for getting the database connection from the data source. Consult database documentation for information on naming conventions.
Globalized	Yes
Bidi supported	No

Note: The **User name used to connect to event data source** property applies only to asynchronous transactional RFC processing.

X509 certificate

This property specifies the X509 certificate to be used as the logon ticket.

Table 115. X509 certificate details

Required	No.
Default	No default value
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), you can provide a value for the X509 certificate.
Globalized	No
Bidi supported	No

Activation specification properties for ALE inbound processing:

Activation specification properties hold the inbound event processing configuration information for a message endpoint. Use the Adapter Connection wizard to set activation specification properties.

Activation specification properties are used during endpoint activation to notify the adapter of eligible event listeners. During inbound processing, the adapter uses these event listeners to receive events before it forwards them to the endpoint.

The following table describes the activation specification properties for ALE inbound processing. A more detailed description of each property is provided in the sections that follow the table.

Table 116. Activation specification properties for ALE inbound processing

Property name	Description
AleFailureCode	The status code for dispatch failure.
AleFailureText	The descriptive text for dispatch failure.
AleSelectiveUpdate	The IDoc Type and MessageType combinations that are to be updated when the adapter is configured to update a standard SAP status code.
AleStatusMsgCode	If required, the message code to use when the adapter posts the ALEAUD Message IDoc (ALEAUD01).
AleSuccessCode	The success status code for Application Document Posted.
AleSuccessText	The descriptive text for successful Application Document Posted.
AleUpdateStatus	Specifies whether an audit trail is required for all message types.
AssuredOnceDelivery	Specifies whether to provide assured once-only delivery for inbound events.
EP_CreateTable	Specifies whether the adapter should create the event recovery table automatically if it does not already exist.
Client	The client number of the SAP system to which the adapter connects.
Codepage	Indicates the numeric identifier of the code page.

Table 116. Activation specification properties for ALE inbound processing (continued)

Property name	Description
EP_SchemaName	The schema that is used for automatically creating the event recovery table.
SncMode	Indicates whether secure network connection mode is used.
EP_DataSource_JNDIName	The JNDI name of the data source that is configured for event recovery.
EP_TableName	The name of the event recovery table.
RfcTracePath	Sets the fully qualified local path to the folder into which the RFC trace files are written.
GatewayHost	The host name of the SAP gateway.
GatewayService	The identifier of the gateway on the gateway host that carries out the RFC services.
ApplicationServerHost	Specifies the IP address or the name of the application server host that the adapter logs on to.
IgnoreIDocPacketErrors	Specifies what the adapter does when it encounters an error while processing the IDoc packet.
Language code	Specifies the Language code in which the adapter logs on to SAP.
Group	An identifier of the name of the group of application server instances that have been defined in transaction SMLG and linked together for logon load balancing.
retryLimit	Specifies the number of times the adapter tries to restart the event listeners.
MessageServerHost	Specifies the name of the host on which the message server is running.
NumberOfListeners	Specifies the number of event listeners that are to be started.
PartnerCharset	Specifies PartnerCharset encoding.
Password	The password of the user account of the adapter on the SAP application server.
EP_Password	The user password for connecting to the database.
RfcProgramID	The remote function call identifier under which the adapter registers in the SAP gateway.
RfcTraceLevel	Specifies the global trace level.
RfcTraceOn	Specifies whether to generate a text file detailing the RFC activity for each event listener.
SAPSystemID	Specifies the system ID of the SAP system for which logon load balancing is allowed.
SncLib	Specifies the path to the library that provides the secure network connection service.
SncMyname	Specifies the name of the secure network connection.
SncPartnername	Specifies the name of the secure network connection partner.
SncQop	Specifies the level of security for the secure network connection.
SystemNumber	The system number of the SAP application server.

Table 116. Activation specification properties for ALE inbound processing (continued)

Property name	Description
retryInterval	Specifies the time interval between attempts to restart the event listeners.
userName	The user account for the adapter on the SAP server.
EP_UserName	The user name for connecting to the database.
X509cert	Specifies the X509 certificate to be used as the logon ticket.

ALE failure code

This property determines how the adapter updates the SAP failure status code after the ALE module has retrieved an IDoc object for event processing.

Table 117. ALE failure code details

Required	Yes if AleUpdateStatus is set to `True`; otherwise, no.
Possible values	`68` `58`
Default	No default value.
Property type	Integer
Usage	Set a value for this property only if you set the value for AleUpdateStatus to `True`. Specify a value of `68` for this property to cause the adapter to update the SAP failure status code after the ALE module has retrieved an IDoc object for event processing. SAP converts this value to 40 (Application Document not created in receiving system). When you set the AleUpdateStatus property to `True`, the adapter updates a standard SAP status code after the adapter retrieves an IDoc object for event processing. An IDoc that is not sent successfully to the endpoint is considered to be a failure. Use the ALE failure code property to specify the code that is used to signify this failure.
Globalized	No
Bidi supported	No

ALE failure text

The property specifies the text that appears when an IDoc is not sent successfully to the endpoint.

Table 118. ALE failure text details

Required	Yes if AleUpdateStatus is set to `True`; otherwise, no.
Default	No default value.
Property type	String
Usage	Use this property only if you set the AleUpdateStatus property to `True`. The length of the text string cannot exceed 70 characters. When you set the AleUpdateStatus property to `True`, the adapter updates a standard SAP status code after the adapter retrieves an IDoc object for event processing. An IDoc that is not sent successfully to the endpoint is considered to be a failure. Use the ALE failure text property to specify the descriptive text that is used to signify this failure.
Example	`ALE Dispatch Failed`

Table 118. ALE failure text details (continued)

Globalized	Yes
Bidi supported	No

ALE selective update

This property specifies which IDoc Type and MessageType combinations are to be updated.

Table 119. ALE selective update details

Required	No
Default	No default value
Property type	String
Usage	You can set values for this property only if AleUpdateStatus is set to `True`. When you set the AleUpdateStatus property to `True`, the adapter updates a standard SAP status code after the adapter retrieves an IDoc object for event processing. Use the ALE selective update property to specify which IDoc Type and MessageType combinations are to be updated. The syntax for this property is: `IDocType: MessageType [;IDocType: MessageType [;...]]` where a slash (/) delimiter separates each IDoc Type and MessageType, and a semicolon (;) delimiter separates entries in a set.
Example	The following example illustrates two sets. In the example, MATMAS03 and DEBMAS03 are the IDocs, and MATMAS and DEBMAS are the message types: `MATMAS03/MATMAS;DEBMAS03/DEBMAS`
Globalized	No
Bidi supported	No

ALE status message code

This property specifies the message code to use when the adapter posts the ALEAUD01 IDoc with message type ALEAUD.

Table 120. ALE status message code details

Required	No
Possible values	For a list of available codes, refer to the SAP table TEDS1.
Default	No default value.
Property type	String
Usage	You can set a value for this property only if AleUpdateStatus has been set to `True`. You must configure this message code in the receiving partner profile on SAP.
Globalized	No
Bidi supported	No

ALE success code

This property specifies the ALE success code for the successful posting of an IDoc.

Table 121. ALE success code details

Required	Yes if AleUpdateStatus is set to `True`; otherwise, no.
Possible values	`52` `53`
Default	No default value.
Property type	String
Usage	Use this property only if you set the AleUpdateStatus property to `True`. When you set the AleUpdateStatus property to `True`, the adapter updates a standard SAP status code after the adapter retrieves an IDoc object for event processing. Use the ALE success code property to specify the code for IDoc posted as 53. After the IDoc is sent to the endpoint, the IDoc status remains as 03 (IDoc posted to port) in SAP. After posting the IDoc, the adapter posts the audit IDoc with the current IDoc number and status as 53. SAP converts the current IDoc status to 41 (Application Document Created in Receiving System).
Globalized	No
Bidi supported	No

ALE success text

This property specifies the text that appears when an application document is posted successfully.

Table 122. ALE success text details

Required	Yes if AleUpdateStatus is set to `True`; otherwise, no.
Default	No default value.
Property type	String
Usage	Use this property only if you set the AleUpdateStatus property to `True`. When you set the AleUpdateStatus property to `True`, the adapter updates a standard SAP status code after the adapter retrieves an IDoc object for event processing. Use the ALE success text property to specify the descriptive text that is used to signify Application Document Posted.
Example	`ALE Dispatch OK`
Globalized	Yes
Bidi supported	No

ALE update status

This property specifies whether an audit trail is required for all message types.

Table 123. ALE update status details

Required	Yes
Possible values	`True` `False`
Default	`False`
Property type	Boolean

Table 123. ALE update status details (continued)

Usage	Set this property to `True` if you want the adapter to update a standard SAP status code after the ALE module has retrieved an IDoc object for event processing. If you set this value to `True`, you must also set following properties: • AleFailureCode • AleSuccessCode • AleFailureText • AleSuccessText.
Globalized	No
Bidi supported	No

Assured once-only delivery (AssuredOnceDelivery)

This property specifies whether to provide assured once-only delivery for inbound events.

Table 124. Assured once-only delivery details

Required	No
Default	`False`
Property type	Boolean
Usage	When this property is set to `True`, the adapter provides assured once-only event delivery, so that each event is delivered only once. A value of `False` does not provide assured once-only event delivery, but provides better performance. When this property is set to `True`, the adapter attempts to store transaction (XID) information in the event store. If it is set to `False`, the adapter does not attempt to store the information. This property is used only if the export component is transactional. If the export component is not transactional, no transaction can be used, regardless of the value of this property.
Globalized	No
Bidi supported	No

Auto create event table

This property determines if the event table is created automatically.

Table 125. Auto create event table details

Required	Yes if **Assured once-only event delivery** is set to `True`; otherwise, no.
Possible values	`True` `False`
Default	`True`
Property type	Boolean
Usage	This property indicates whether the adapter should create the event recovery table automatically if it does not already exist. If you specify a value of `True` to automatically create the table, you must specify information about the event table (such as the event recovery table name). The value that is provided in Event recovery table name property is used to create the table.
Globalized	No

Table 125. Auto create event table details (continued)

Bidi supported	No

Client

This property is the client number of the SAP system to which the adapter connects.

Table 126. Client details

Required	Yes
Possible values	You can enter a range of values from `000` to `999`.
Default	`100`
Property type	Integer
Usage	When an application attempts to log on to the SAP server, the application must have a Client number associated with it. The Client property value identifies the client (the adapter) that is attempting to log onto the SAP server.
Globalized	No
Bidi supported	No

Codepage number

The numeric identifier of the code page.

Table 127. Codepage number details

Required	No
Possible values	You can enter a range of values from `0000` to `9999`. For a full listing of languages and associated code page numbers that are supported by SAP, see SAP Note 7360.
Default	The default value for this property is conditionally determined by the value set for the **Language code** property.
Property type	Integer
Usage	The value that is assigned to the **Codepage number** defines the code page to be used and has a one-to-one relationship with the value set for the **Language code** property. The **Codepage number** establishes a connection to the appropriate language. Each language code value has a code page number value associated with it. For example, the language code for English is EN. If you select EN (English) as your language code, the code page number is automatically set to the numeric value that is associated with EN (English). The SAP code page number for EN (English) is `1100`.
Example	If **Language code** is set to `JA` (Japanese), **Codepage number** is set to `8000`.
Globalized	No
Bidi supported	No

Database schema name

This property is the schema that is used to create the event recovery table automatically.

Table 128. Database schema name details

Required	No
Default	No default value.
Property type	String
Usage	Specifies the schema name for the database used by the adapter's event persistence feature.
Example	`ALE_SCHEMA`
Globalized	Yes
Bidi supported	No

Enable Secure Network Connection

This property indicates whether secure network connection mode is enabled.

Table 129. Enable Secure Network Connection details

Required	No
Possible values	`0` (off) `1` (on)
Default	`0`
Property type	String
Usage	Set the value to 1 (on) if you want to use secure network connection. If you set this value to 1, you must also set following properties: • "Secure Network Connection library path" on page 1383 • "Secure Network Connection name" on page 1384 • "Secure Network Connection partner" on page 1384 • "Secure Network Connection security level" on page 1384.
Globalized	No
Bidi supported	No

Event recovery data source (JNDI) name

This property is the JNDI name of the data source that is configured for event recovery.

Table 130. Event recovery data source (JNDI) name details

Required	Yes
Default	No default value.
Property type	String
Usage	Used in event recovery processing. The data source must be created in WebSphere Message Broker. The adapter uses the data source for persisting the event state.
Example	`jdbc/DB2`
Globalized	No
Bidi supported	No

Event recovery table name

This property is the name of the event recovery table.

Table 131. Event recovery table name details

Required	Yes
Default	No default value.
Property type	String
Usage	Used in event recovery processing. Consult database documentation for information on naming conventions. Configure a separate event recovery table for each endpoint. The same data source can be used to hold all of the event recovery tables.
Example	`EVENT_TABLE`
Globalized	No
Bidi supported	No

Folder for RFC trace files

This property sets the fully qualified local path to the folder in which to write RFC trace files.

Table 132. Folder for RFC trace files details

Required	No
Default	No default value
Property type	String
Usage	Identifies the fully qualified local path into which RFC trace files are written. If **RFC trace on** is set to False (not selected), you are not permitted to set a value in the **Folder for RFC trace files** property.
Example	`c:\temp\rfcTraceDir`
Globalized	Yes
Bidi supported	No

Gateway host

This property is the Gateway host name. Enter either the IP address or the name of the Gateway host. Consult your SAP administrator for information on the Gateway host name.

Table 133. Gateway host details

Required	Yes
Default	No default value
Property type	String
Usage	This property is the host name of the SAP gateway. The gateway enables communication between work processes on the SAP system and external programs. The host identified is used as the gateway for the resource adapter. Maximum length of 20 characters. If the computer name is longer than 20 characters, define a symbolic name in the THOSTS table.

Table 133. Gateway host details (continued)

Globalized	No
Bidi supported	No

Gateway service

This property is the identifier of the gateway on the gateway host that carries out the RFC services.

Table 134. Gateway service details

Required	Yes
Default	`sapgw00`
Property type	String
Usage	These services enable communication between work processes on the SAP server and external programs. The service typically has the format of `sapgw00`, where 00 is the SAP system number. Maximum of 20 characters.
Globalized	No
Bidi supported	No

Host name

Specifies the IP address or the name of the application server host that the adapter logs on to.

Table 135. Host name details

Required	Yes (when load balancing is not used).
Default	No default value
Property type	String
Usage	When the adapter is configured to run without load balancing, this property specifies the IP address or the name of the application server that the adapter logs on to.
Example	`sapServer`
Globalized	No
Bidi supported	No

Ignore IDoc packet errors

This property determines whether IDoc packet errors are to be ignored.

Table 136. Ignore IDOC packet errors details

Required	No
Possible values	`True` `False`
Default	`False`
Property type	Boolean

Table 136. Ignore IDOC packet errors details (continued)

Usage	If the adapter encounters an error while processing the IDoc packet, it can behave in two different ways. • When this property is set to `False`, the adapter stops processing further IDocs in that packet and reports an error to the SAP system. • When this property is set to `True`, the adapter logs an error and continues to process the rest of the IDocs in that packet. The status of the transaction is marked as `INPROGRESS`. The adapter log displays the IDoc numbers that failed and you need to resubmit those individual IDocs separately. You need to manually maintain these records in the event recovery table. This property is not used for single IDocs and for non-split IDoc packets.
Globalized	No
Bidi supported	No

Language code

This property specifies the Language code in which the adapter logs on.

Table 137. Language code details

Required	Yes
Possible values	For a full listing of languages and associated code page numbers that are supported by SAP, see SAP Note 7360.
Default	The default value for the **Language code** property is based on the system locale.
Property type	String
Usage	Each of the supported languages is preceded by a two-character language code. The language itself is displayed in parentheses. If you enter a language code manually, you do not need to enter the language in parentheses. The language codes that are listed represent the SAP default set of 41 languages for non-Unicode systems plus Arabic. The value that you select determines the value of the **Codepage number** property.
Example	If the system locale is English, the value for this property is `EN (English)`.
Globalized	No
Bidi supported	No

Logon group name

This property is an identifier for the name of the group of application server instances that have been defined in transaction SMLG and linked together for logon load balancing.

Table 138. Logon group details

Required	Yes (if load balancing is used)
Possible values	Consult SAP documentation for information on creating Logon groups and on calling transaction SMLG.
Default	No default value
Property type	String

Table 138. Logon group details (continued)

Usage	When the adapter is configured for load balancing, this property represents the name of the group of application server instances that have been defined in transaction SMLG and linked together for logon load balancing. Logon load balancing allows for the dynamic distribution of logon connections to application server instances. Maximum of 20 characters. On most SAP systems, the SPACE logon group is reserved by SAP.
Globalized	No
Bidi supported	No

Maximum number of retries in case of system connection failure

This property specifies the number of times the adapter tries to restart the event listeners.

Table 139. Maximum number of retries in case of system failure details

Required	Yes
Default	`0`
Property type	Integer
Usage	When the adapter encounters an error related to the inbound connection (if the SAP application is down for example), this property specifies the number of times the adapter tries to restart the event listeners. A value of 0 indicates an infinite number of retries. **Note:** Configure the **Time between retries in case of system connection failure (milliseconds)** appropriately when retrying infinitely. For each retry attempt, the adapter waits based on the time interval specified in the **Time between retries in case of system connection failure (milliseconds)**. **Note:** If all the retry attempts fail, the adapter logs relevant messages and CEI events and stops attempting to recover the event listener. If you reach this point, you might have to restart the application manually.
Globalized	No
Bidi supported	No

Message server host

This property specifies the name of the host on which the message server is running.

Table 140. Message server host details

Required	Yes (if load balancing is used)
Default	No default value
Property type	String
Usage	This property specifies the name of the host that will inform all the servers (instances) belonging to this SAP system of the existence of the other servers to be used for load balancing. The message server host contains the information about load balancing for RFC clients so that an RFC client can be directed to an appropriate application server.
Example	`SAPERP05`
Globalized	No
Bidi supported	No

Number of listeners

This property specifies the number of listeners that are started by an event.

Table 141. Number of listeners details

Required	No
Default	1
Property type	Integer
Usage	For event sequencing, this property should be set to 1. To improve adapter performance, you can increase the number of listeners.
Globalized	No
Bidi supported	No

Partner character set

This property specifies the partner character set encoding.

Table 142. Partner character set details

Required	No
Default	UTF-8
Property type	String
Usage	When an encoding is specified, it is used; otherwise, the default encoding is used.
Globalized	No
Bidi supported	No

Password

This property is the password of the user account of the adapter on the SAP application server.

Table 143. Password details

Required	Yes
Default	No default value
Property type	String
Usage	The restrictions on the password depend on the version of SAP Web Application Server. • For SAP Web Application Server version 6.40 or earlier, the password: – Must be uppercase – Must be 8 characters in length • For versions of SAP Web Application Server later than 6.40, the password: – Is not case-sensitive – Can be up to 40 characters in length
Globalized	No
Bidi supported	Yes

Password used to connect to event data source

This property is the user password for connecting to the database.

Table 144. Password to connect to event data source details

Required	Yes
Default	No default value.
Property type	String
Usage	This property specifies the password used by event-persistence processing to obtain the database connection from the data source.
Globalized	Yes
Bidi supported	No

RFC program ID

This property is the program identifier under which the adapter registers in the SAP gateway.

Table 145. RFC program ID details

Required	Yes
Possible values	Use the SAP transaction SM59 (Display and Maintain RFC Destinations) to see a list of available RFC program IDs.
Default	No default value.
Property type	String
Usage	The adapter registers with the gateway so that listener threads can process events from RFC-enabled functions. This value must match the program ID registered in the SAP application. The maximum length is 64 characters.
Globalized	No
Bidi supported	No

RFC trace level

This property specifies the global trace level.

Table 146. RFC trace level details

Required	No
Possible values	`1` - This is the default RFC trace level. When specified, SAP JCo Java API logging occurs. `3` - When specified, SAP JCo JNI API logging occurs. `5` - When specified, error diagnostic logging occurs.
Default	`1`
Property type	Integer
Usage	If **RFC trace on** is set to `False` (not selected), you cannot set a value in the **RFC trace level** property.
Globalized	No
Bidi supported	No

RFC trace on

This property specifies whether to generate a text file detailing the RFC activity for each event listener.

Table 147. RFC trace on details

Required	No
Possible values	`True` `False`
Default	`False`
Property type	Boolean
Usage	A value of `True` activates tracing, which generates a text file. This file is created in the directory in which the adapter process was started. The file has a prefix of `rfx` and a file type of trc (for example, rfc03912_02220.trc). Use these text files in a development environment only, because the files can grow rapidly. If **RFC trace on** is set to `False` (not selected), you cannot set values in the **Folder for RFC trace files** or **RFC trace level** properties.
Example	Examples of the information in the file are RfcCall FUNCTION BAPI_CUSTOMER_GETLIST, followed by the information for the parameters in the interface, or RFC Info rfctable, followed by the data from one of the interface tables. The trace file is created in the directory where the adapter process has been started. The trace file has a .trc file extension and the file name will start with the letters `rfc` followed by a unique identifier. For example, `rfc03912_02220.trc`.
Globalized	No
Bidi supported	No

SAP system ID

This property specifies the system ID of the SAP system for which logon load balancing is allowed.

Table 148. SAP system ID details

Required	Yes (when load balancing is used)
Default	No default value
Property type	String
Usage	Value must be three characters
Example	`DYL`
Globalized	No
Bidi supported	No

Secure Network Connection library path

This property specifies the path to the library that provides the secure network connection service.

Table 149. Secure Network Connection library path details

Required	Yes, if SncMode is set to 1; no otherwise.

Table 149. Secure Network Connection library path details (continued)

Default	No default value
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), specify the path to the library that provides the service.
Example	`/WINDOWS/system32/gssapi32.dll`
Globalized	No
Bidi supported	No

Secure Network Connection name

This property specifies the name of the secure network connection.

Table 150. Secure Network Connection name details

Required	Yes, if SncMode is set to 1; no otherwise.
Default	No default value
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), specify a name for the connection.
Example	`DOMAINNAME/USERNAME`
Globalized	No
Bidi supported	No

Secure Network Connection partner

This property specifies the name of the secure network connection partner.

Table 151. Secure Network Connection partner details

Required	Yes, if SncMode is set to 1; no otherwise.
Default	No default value
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), specify a name for the connection partner.
Example	`CN=sap00.saperpdev, OU=Adapter, O=IBM, C=US`
Globalized	No
Bidi supported	No

Secure Network Connection security level

This property specifies the level of security for the secure network connection.

Table 152. Secure Network Connection security level details

Required	Yes, if SncMode is set to 1; no otherwise.

Table 152. Secure Network Connection security level details (continued)

Possible values	1 (Authentication only) 2 (Integrity protection) 3 (Privacy protection) 8 (Use the value from snc/data_protection/use on the application server) 9 (Use the value from snc/data_protection/max on the application server)
Default	3 (Privacy protection)
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), specify a value to indicate the level of security for the connection.
Globalized	No
Bidi supported	No

System number

This property is the system number of the SAP application server.

Table 153. System number details

Required	Yes
Possible values	You can enter a range of values from 00 to 99.
Default	00
Property type	Integer
Usage	The system number further identifies the Gateway service.
Globalized	No
Bidi supported	No

Time between retries in case of system connection failure (milliseconds)

This property specifies the time interval between attempts to restart the event listeners.

Table 154. Time between retries in case of system connection failure details

Required	Yes
Default	60000
Unit of measure	Milliseconds
Property type	Integer
Usage	When the adapter encounters an error related to the inbound connection, this property specifies the time interval the adapter waits in between attempts to restart the event listeners.
Globalized	No
Bidi supported	No

User name

This property is the user account for the adapter on the SAP server.

Table 155. User name details

Required	Yes
Default	No default value
Property type	String
Usage	Maximum length of 12 characters. The user name is not case sensitive. It is recommended that you set up a CPIC user account in the SAP application and that you give this account the necessary privileges to manipulate the data required by the business objects supported by the adapter. For example, if the adapter must perform certain SAP business transactions, the adapter's account in the SAP application must have the permissions set to allow it to perform these transactions.
Example	`SapUser`
Globalized	Yes
Bidi supported	Yes

User name used to connect to event data source

This property is the user name for connecting to the database.

Table 156. User name to connect to event data source details

Required	Yes
Default	No default value.
Property type	String
Usage	User name used by event persistence for getting the database connection from the data source. Consult database documentation for information on naming conventions.
Globalized	Yes
Bidi supported	No

X509 certificate

This property specifies the X509 certificate to be used as the logon ticket.

Table 157. X509 certificate details

Required	No.
Default	No default value
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), you can provide a value for the X509 certificate.
Globalized	No
Bidi supported	No

Activation specification properties for Advanced event processing:

Activation specification properties are properties that hold the inbound event-processing configuration information for a message endpoint. Use the Adapter Connection wizard to set activation specification properties.

Activation specification properties are used during endpoint activation to notify the adapter of eligible event listeners. During inbound processing, the adapter uses these event listeners to receive events before it forwards them to the endpoint.

The following table lists the activation specification properties for Advanced event inbound processing. A complete description of each property is provided in the sections that follow the table.

Table 158. Activation specification properties for Advanced event processing

Property name	Description
AssuredOnceDelivery	Specifies whether to provide assured once-only delivery for inbound events.
Client	The client number of the SAP system to which the adapter connects.
Codepage	Indicates the numeric identifier of the code page.
SncMode	Indicates whether secure network connection mode is used.
DeliveryType	Determines the order in which events are delivered by the adapter to the export component
Event type filter	A delimited list of event types that the WebSphere Adapter for SAP Software should deliver.
RfcTracePath	Sets the fully qualified local path to the folder into which the RFC trace files are written.
GatewayHost	The host name of the SAP gateway.
GatewayService	The identifier of the gateway on the gateway host that carries out the RFC services.
ApplicationServerHost	Specifies the IP address or the name of the application server host that the adapter logs on to.
Language code	Specifies the Language code in which the adapter logs on to SAP.
Group	An identifier of the name of the group of application server instances that have been defined in transaction SMLG and linked together for logon load balancing.
PollQuantity	The number of events that the adapter delivers to the export during each poll period.
RetryLimit	The number of times that the adapter tries to reestablish an inbound connection after an error.
MessageServerHost	Specifies the name of the host on which the message server is running.
PartnerCharset	Specifies PartnerCharset encoding.
Password	The password of the user account of the adapter on the SAP application server.
RfcTraceLevel	Specifies the global trace level.
RfcTraceOn	Specifies whether to generate a text file detailing the RFC activity for each event listener.

Table 158. Activation specification properties for Advanced event processing (continued)

Property name	Description
SAPSystemID	Specifies the system ID of the SAP system for which logon load balancing is allowed.
SncLib	Specifies the path to the library that provides the secure network connection service.
SncMyname	Specifies the name of the secure network connection.
SncPartnername	Specifies the name of the secure network connection partner.
SncQop	Specifies the level of security for the secure network connection.
StopPollingOnError	Specifies whether the adapter stops polling for events when it encounters an error during polling.
SystemNumber	The system number of the SAP application server.
PollPeriod	The length of time that the adapter waits between polling periods.
RetryInterval	The length of time that the adapter waits between attempts to establish a new connection after an error during inbound operations.
userName	The user account for the adapter on the SAP server.
X509cert	Specifies the X509 certificate to be used as the logon ticket.

Assured once-only delivery (AssuredOnceDelivery)

This property specifies whether to provide assured once-only delivery for inbound events.

Table 159. Assured once-only delivery details

Required	No
Default	`False`
Property type	Boolean
Usage	When this property is set to `True`, the adapter provides assured once-only event delivery, so that each event is delivered only once. A value of `False` does not provide assured once-only event delivery, but provides better performance. When this property is set to `True`, the adapter attempts to store transaction (XID) information in the event store. If it is set to `False`, the adapter does not attempt to store the information. This property is used only if the export component is transactional. If the export component is not transactional, no transaction can be used, regardless of the value of this property.
Globalized	No
Bidi supported	No

Client

This property is the client number of the SAP system to which the adapter connects.

Table 160. Client details

Required	Yes
Possible values	You can enter a range of values from `000` to `999`.
Default	`100`
Property type	Integer
Usage	When an application attempts to log on to the SAP server, the application must have a Client number associated with it. The Client property value identifies the client (the adapter) that is attempting to log onto the SAP server.
Globalized	No
Bidi supported	No

Codepage number

The numeric identifier of the code page.

Table 161. Codepage number details

Required	No
Possible values	You can enter a range of values from `0000` to `9999`. For a full listing of languages and associated code page numbers that are supported by SAP, see SAP Note 7360.
Default	The default value for this property is conditionally determined by the value set for the **Language code** property.
Property type	Integer
Usage	The value that is assigned to the **Codepage number** defines the code page to be used and has a one-to-one relationship with the value set for the **Language code** property. The **Codepage number** establishes a connection to the appropriate language. Each language code value has a code page number value associated with it. For example, the language code for English is EN. If you select EN (English) as your language code, the code page number is automatically set to the numeric value that is associated with EN (English). The SAP code page number for EN (English) is `1100`.
Example	If **Language code** is set to `JA` (Japanese), **Codepage number** is set to `8000`.
Globalized	No
Bidi supported	No

Delivery type (DeliveryType)

This property specifies the order in which events are delivered by the adapter to the export component.

Table 162. Delivery type details

Required	No
Possible values	`ORDERED` `UNORDERED`
Default	`ORDERED`
Property type	String

Table 162. Delivery type details (continued)

Usage	The following values are supported: • `ORDERED`: The adapter delivers events to the export component one at a time. • `UNORDERED`: The adapter delivers all events to the export component at once.
Globalized	No
Bidi supported	No

Enable Secure Network Connection

This property indicates whether secure network connection mode is enabled.

Table 163. Enable Secure Network Connection details

Required	No
Possible values	`0` (off) `1` (on)
Default	`0`
Property type	String
Usage	Set the value to 1 (on) if you want to use secure network connection. If you set this value to 1, you must also set following properties: • "Secure Network Connection library path" on page 1396 • "Secure Network Connection name" on page 1396 • "Secure Network Connection partner" on page 1396 • "Secure Network Connection security level" on page 1397
Globalized	No
Bidi supported	No

Event type filter

This property provides a delimited list of business object types for which the adapter should deliver events.

Table 164. Event type filter details

Required	No
Default	`Null`
Property type	String
Usage	The adapter uses the delimited list as a filter, delivering events for only those business object types that are contained in the list. If the list is empty (null), the adapter does not apply filtering, and delivers events for all business object types.
Globalized	No
Bidi supported	No

Folder for RFC trace files

This property sets the fully qualified local path to the folder in which to write RFC trace files.

Table 165. Folder for RFC trace files details

Required	No
Default	No default value
Property type	String
Usage	Identifies the fully qualified local path into which RFC trace files are written. If **RFC trace on** is set to False (not selected), you are not permitted to set a value in the **Folder for RFC trace files** property.
Example	`c:\temp\rfcTraceDir`
Globalized	Yes
Bidi supported	No

Gateway host

This property is the Gateway host name. Enter either the IP address or the name of the Gateway host. Consult your SAP administrator for information on the Gateway host name.

Table 166. Gateway host details

Required	Yes
Default	No default value
Property type	String
Usage	This property is the host name of the SAP gateway. The gateway enables communication between work processes on the SAP system and external programs. The host identified is used as the gateway for the resource adapter. Maximum length of 20 characters. If the computer name is longer than 20 characters, define a symbolic name in the THOSTS table.
Globalized	No
Bidi supported	No

Gateway service

This property is the identifier of the gateway on the gateway host that carries out the RFC services.

Table 167. Gateway service details

Required	Yes
Default	`sapgw00`
Property type	String
Usage	These services enable communication between work processes on the SAP server and external programs. The service typically has the format of `sapgw00`, where 00 is the SAP system number. Maximum of 20 characters.
Globalized	No
Bidi supported	No

Host name

Specifies the IP address or the name of the application server host that the adapter logs on to.

Table 168. Host name details

Required	Yes (when load balancing is not used).
Default	No default value
Property type	String
Usage	When the adapter is configured to run without load balancing, this property specifies the IP address or the name of the application server that the adapter logs on to.
Example	`sapServer`
Globalized	No
Bidi supported	No

Language code

This property specifies the Language code in which the adapter logs on.

Table 169. Language code details

Required	Yes
Possible values	For a full listing of languages and associated code page numbers that are supported by SAP, see SAP Note 7360.
Default	The default value for the **Language code** property is based on the system locale.
Property type	String
Usage	Each of the supported languages is preceded by a two-character language code. The language itself is displayed in parentheses. If you enter a language code manually, you do not need to enter the language in parentheses. The language codes that are listed represent the SAP default set of 41 languages for non-Unicode systems plus Arabic. The value that you select determines the value of the **Codepage number** property.
Example	If the system locale is English, the value for this property is `EN (English)`.
Globalized	No
Bidi supported	No

Logon group name

This property is an identifier for the name of the group of application server instances that have been defined in transaction SMLG and linked together for logon load balancing.

Table 170. Logon group details

Required	Yes (if load balancing is used)
Possible values	Consult SAP documentation for information on creating Logon groups and on calling transaction SMLG.
Default	No default value
Property type	String

Table 170. Logon group details (continued)

Usage	When the adapter is configured for load balancing, this property represents the name of the group of application server instances that have been defined in transaction SMLG and linked together for logon load balancing. Logon load balancing allows for the dynamic distribution of logon connections to application server instances. Maximum of 20 characters. On most SAP systems, the SPACE logon group is reserved by SAP.
Globalized	No
Bidi supported	No

Maximum number of events collected during each poll

This property specifies the number of events that the adapter delivers to the export component during each poll period.

Table 171. Maximum number of events collected during each poll details

Required	Yes
Default	`10`
Property type	Integer
Usage	This value must be greater than 0
Globalized	No
Bidi supported	No

Maximum number of retries in case of system connection failure

This property specifies the number of times that the adapter tries to reestablish an inbound connection.

Table 172. Maximum number of retries in case of system connection failure details

Required	No
Possible values	Positive integers
Default	`0`
Property type	Integer
Usage	Only positive values are valid. When the adapter encounters an error that is related to the inbound connection, this property specifies the number of times that the adapter tries to restart the connection. A value of `0` indicates an infinite number of retries.
Globalized	No
Bidi supported	No

Message server host

This property specifies the name of the host on which the message server is running.

Table 173. Message server host details

Required	Yes (if load balancing is used)

Table 173. Message server host details (continued)

Default	No default value
Property type	String
Usage	This property specifies the name of the host that will inform all the servers (instances) belonging to this SAP system of the existence of the other servers to be used for load balancing. The message server host contains the information about load balancing for RFC clients so that an RFC client can be directed to an appropriate application server.
Example	`SAPERP05`
Globalized	No
Bidi supported	No

Partner character set

This property specifies the partner character set encoding.

Table 174. Partner character set details

Required	No
Default	UTF-8
Property type	String
Usage	When an encoding is specified, it is used; otherwise, the default encoding is used.
Globalized	No
Bidi supported	No

Password

This property is the password of the user account of the adapter on the SAP application server.

Table 175. Password details

Required	Yes
Default	No default value
Property type	String
Usage	The restrictions on the password depend on the version of SAP Web Application Server. • For SAP Web Application Server version 6.40 or earlier, the password: – Must be uppercase – Must be 8 characters in length • For versions of SAP Web Application Server later than 6.40, the password: – Is not case-sensitive – Can be up to 40 characters in length
Globalized	No
Bidi supported	Yes

RFC trace level

This property specifies the global trace level.

Table 176. RFC trace level details

Required	No
Possible values	1 - This is the default RFC trace level. When specified, SAP JCo Java API logging occurs. 3 - When specified, SAP JCo JNI API logging occurs. 5 - When specified, error diagnostic logging occurs.
Default	1
Property type	Integer
Usage	If **RFC trace on** is set to `False` (not selected), you cannot set a value in the **RFC trace level** property.
Globalized	No
Bidi supported	No

RFC trace on

This property specifies whether to generate a text file detailing the RFC activity for each event listener.

Table 177. RFC trace on details

Required	No
Possible values	`True` `False`
Default	`False`
Property type	Boolean
Usage	A value of `True` activates tracing, which generates a text file. This file is created in the directory in which the adapter process was started. The file has a prefix of `rfx` and a file type of trc (for example, rfc03912_02220.trc). Use these text files in a development environment only, because the files can grow rapidly. If **RFC trace on** is set to `False` (not selected), you cannot set values in the **Folder for RFC trace files** or **RFC trace level** properties.
Example	Examples of the information in the file are RfcCall FUNCTION BAPI_CUSTOMER_GETLIST, followed by the information for the parameters in the interface, or RFC Info rfctable, followed by the data from one of the interface tables. The trace file is created in the directory where the adapter process has been started. The trace file has a .trc file extension and the file name will start with the letters `rfc` followed by a unique identifier. For example, `rfc03912_02220.trc`.
Globalized	No
Bidi supported	No

SAP system ID

This property specifies the system ID of the SAP system for which logon load balancing is allowed.

Table 178. SAP system ID details

Required	Yes (when load balancing is used)
Default	No default value
Property type	String
Usage	Value must be three characters
Example	`DYL`
Globalized	No
Bidi supported	No

Secure Network Connection library path

This property specifies the path to the library that provides the secure network connection service.

Table 179. Secure Network Connection library path details

Required	Yes, if SncMode is set to 1; no otherwise.
Default	No default value
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), specify the path to the library that provides the service.
Example	`/WINDOWS/system32/gssapi32.dll`
Globalized	No
Bidi supported	No

Secure Network Connection name

This property specifies the name of the secure network connection.

Table 180. Secure Network Connection name details

Required	Yes, if SncMode is set to 1; no otherwise.
Default	No default value
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), specify a name for the connection.
Example	`DOMAINNAME/USERNAME`
Globalized	No
Bidi supported	No

Secure Network Connection partner

This property specifies the name of the secure network connection partner.

Table 181. Secure Network Connection partner details

Required	Yes, if SncMode is set to 1; no otherwise.
Default	No default value
Property type	String

Table 181. Secure Network Connection partner details (continued)

Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), specify a name for the connection partner.
Example	`CN=sap00.saperpdev, OU=Adapter, O=IBM, C=US`
Globalized	No
Bidi supported	No

Secure Network Connection security level

This property specifies the level of security for the secure network connection.

Table 182. Secure Network Connection security level details

Required	Yes, if SncMode is set to 1; no otherwise.
Possible values	`1` (Authentication only) `2` (Integrity protection) `3` (Privacy protection) `8` (Use the value from `snc/data_protection/use` on the application server) `9` (Use the value from `snc/data_protection/max` on the application server)
Default	`3` (Privacy protection)
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), specify a value to indicate the level of security for the connection.
Globalized	No
Bidi supported	No

Stop the adapter when an error is encountered while polling (StopPollingOnError)

This property specifies whether the adapter stops polling for events when it encounters an error during polling.

Table 183. Stop the adapter when an error is encountered while polling details

Required	No
Possible values	`True` `False`
Default	`False`
Property type	Boolean
Usage	If this property is set to `True`, the adapter stops polling when it encounters an error. If this property is set to `False`, the adapter logs an exception when it encounters an error during polling and continues polling.
Globalized	No
Bidi supported	No

System number

This property is the system number of the SAP application server.

Table 184. System number details

Required	Yes
Possible values	You can enter a range of values from `00` to `99`.
Default	`00`
Property type	Integer
Usage	The system number further identifies the Gateway service.
Globalized	No
Bidi supported	No

Time between polling for events (milliseconds)

This property specifies the length of time that the adapter waits between polling periods.

Table 185. Time between polling for events (milliseconds)

Required	Yes
Possible values	Integers greater than or equal to `0`.
Default	`2000`
Unit of measure	Milliseconds
Property type	Integer
Usage	The time interval between polling events is established at a fixed rate, which means that if running the poll cycle is delayed for any reason (for example, if a prior poll cycle takes longer than expected to complete), the next poll cycle occurs immediately to make up for the time that is lost because of the delay.
Globalized	No
Bidi supported	No

Time between retries in case of system connection failure (milliseconds)

This property specifies the time interval between attempts to reestablish an inbound connection.

Table 186. Time between retries in case of system connection failure details

Required	Yes
Default	`60000`
Unit of measure	Milliseconds
Property type	Integer
Usage	When the adapter encounters an error that is related to the inbound connection, this property specifies the time interval that the adapter waits in between attempts to reestablish an inbound connection.
Globalized	No
Bidi supported	No

User name

This property is the user account for the adapter on the SAP server.

Table 187. User name details

Required	Yes
Default	No default value
Property type	String
Usage	Maximum length of 12 characters. The user name is not case sensitive. It is recommended that you set up a CPIC user account in the SAP application and that you give this account the necessary privileges to manipulate the data required by the business objects supported by the adapter. For example, if the adapter must perform certain SAP business transactions, the adapter's account in the SAP application must have the permissions set to allow it to perform these transactions.
Example	`SapUser`
Globalized	Yes
Bidi supported	Yes

X509 certificate

This property specifies the X509 certificate to be used as the logon ticket.

Table 188. X509 certificate details

Required	No.
Default	No default value
Property type	String
Usage	If the SncMode property is set to 1 (indicating that you are using a secure network connection), you can provide a value for the X509 certificate.
Globalized	No
Bidi supported	No

Interaction specification properties (SAP):

An interaction is an operation. Interaction specification properties control how the operation is run.

The Adapter Connection wizard sets the interaction specification properties when you configure the adapter. Typically, you do not need to change these properties. However, you can change some properties for outbound operations. For example, you can increase the value of the interaction specification property that specifies the maximum number of hits for the discovery to be returned by a `RetrieveAll` operation, if your `RetrieveAll` operations do not return complete information.

The following table lists the interaction specification properties that you can set.

Table 189. Interaction specification properties for Adapter for SAP Software

Property name	Description
"Custom retrieve function name" on page 1400	The name of a custom function to be used by the Query interface to SAP Software to retrieve data from an SAP table.

Table 189. Interaction specification properties for Adapter for SAP Software (continued)

Property name	Description
"Function name"	The function name for a specific SAP interface.
"Ignore errors in BAPI return" on page 1401	Specifies if errors in BAPI return are ignored.
"Maximum number of hits for the discovery" on page 1402	The maximum number of result sets to return during a `RetrieveAll` operation.
"Select the queue name" on page 1402	The name of a customer-defined queue on the SAP server.

Custom retrieve function name

For the Query interface for SAP Software, this property specifies the name of a custom function that must be used to retrieve data from an SAP table.

Table 190. Custom retrieve function name details

Required	No
Default	No default value
Property type	String
Usage	This property applies to Query interface for SAP Software only. On non-Unicode systems, the default function used to retrieve data from SAP tables (RFC_READ_TABLE) might produce an exception. To avoid the problem, you can create another function on the SAP server and indicate, during configuration, that the adapter must use this custom function to retrieve data. This property specifies the name of the custom function. You must create the function on the SAP server before you specify this property on the Adapter Connection wizard. Follow the steps listed in SAP note 758278 to make a copy of RFC_READ_TABLE and modify the copy in line with the note.
Globalized	No
Bidi supported	No

Function name

The `functionName` interaction specification property controls the interaction by associating operations with the correct interface.

Table 191. Function name details

Required	Yes
Possible values	`True` `False`
Default	Null
Property type	String

Table 191. Function name details (continued)

Usage	The BAPI/RFC supports the following values for the functionName interaction specification property:
	`WBIInteractionSpec.CREATE`
	`WBIInteractionSpec.UPDATE`
	`WBIInteractionSpec.RETRIEVE`
	`WBIInteractionSpec.DELETE`
	
	The BAPI result set supports the following value for the functionName interaction specification property:
	
	`WBIInteractionSpec.RETRIEVEALL`
	
	The ALE outbound interface supports the following value:
	
	`WBIInteractionSpec.EXECUTE`
	
	The ALE inbound interface supports the following values for the functionName interaction specification property:
	
	`WBIInteractionSpec.CREATE`
	`WBIInteractionSpec.UPDATE`
	`WBIInteractionSpec.RETRIEVE`
	`WBIInteractionSpec.DELETE`
	
	The Query interface for SAP software (QISS) interface supports the following values for the functionName interaction specification property:
	• `WBIInteractionSpec.EXISTS`
	Throws the exceptions `NotExistsException` and `QISSQueryFailedException`
	• `WBIInteractionSpec.RETRIEVEALL`
	Throws the exceptions `QISSQueryFailedException`
	
	The Advanced event processing interface for inbound processing supports the following values for the functionName interaction specification property:
	
	`WBIInteractionSpec.CREATE`
	`WBIInteractionSpec.UPDATE`
	`WBIInteractionSpec.DELETE`
	
	The Advanced event processing interface for outbound processing supports the following values for the functionName interaction specification property:
	
	`WBIInteractionSpec.CREATE`
	`WBIInteractionSpec.UPDATE`
	`WBIInteractionSpec.RETRIEVE`
	`WBIInteractionSpec.DELETE`

Globalized	No
Bidi supported	No

Ignore errors in BAPI return

This property indicates whether to ignore errors that are specified in a BAPI return operation. The return structure can be data or a table.

Table 192. Ignore errors in BAPI return details

Required	No

Table 192. Ignore errors in BAPI return details (continued)

Possible values	`True` `False`
Default	`False`
Property type	Boolean
Usage	This property applies to BAPI outbound synchronous RFC processing only. When you set this property to `True`, the Adapter for SAP Software ignores the checking of error codes in the BAPI RETURN structure after BAPI has run, and returns this structure to the user as it is. The RETURN structure is part of every BAPI and contains the status of the BAPI execution. If you accept the default value of `False`, the Adapter for SAP Software processes the RETURN structure and throws an exception if an error code is found.
Globalized	No
Bidi supported	No

Maximum number of hits for the discovery

For the Query interface for SAP Software, this property specifies the maximum number of result sets, which represents data for each row that is retrieved from a table through a RetrieveAll operation.

Table 193. Result set limit details

Required	Yes
Default	`100`
Property type	Integer
Usage	This property applies to Query interface for SAP Software only. If the number of hits in the table on the SAP server exceeds the value of the ResultSetLimit property, the adapter returns the error: `MatchesExceededLimitException`. The adapter uses this property to help avoid out-of-memory issues.
Globalized	No
Bidi supported	No

Select the queue name

For BAPI outbound processing, when Asynchronous queued RFC is selected, this property specifies the name of a queue on the SAP server to which BAPIs will be delivered.

Table 194. Select the queue name details

Required	No
Default	The first queue defined on the SAP server. If no queue is defined on the SAP server, no default value exists.
Property type	String
Usage	This property applies to BAPI outbound asynchronous queued RFC processing only. When you want to deliver BAPI calls to a queue on the SAP server, you must specify the name of the queue. During configuration, you select an existing queue from a drop-down list. If no queues exist on the SAP server, you can type the name of a queue.

Table 194. Select the queue name details (continued)

Globalized	No
Bidi supported	No

WebSphere Adapter for Siebel properties

Reference information to refer to when you connect to a Siebel application.

- "Business object information (Siebel)"
- "Supported data operations (Siebel)"
- "Naming conventions for business objects representing Siebel business services" on page 1404
- "Adding external software dependencies for Siebel" on page 291
- "Configuration properties for the WebSphere Adapter for Siebel Business Applications" on page 1407
- "Siebel connection properties for the Adapter Connection wizard" on page 1407
- "Resource adapter properties (Siebel)" on page 1414
- "Managed connection factory properties (Siebel)" on page 1415
- "Activation specification properties (Siebel)" on page 1418

Business object information (Siebel)

A business object is a structure that contains application-specific information (metadata) about how the adapter should process the business object as well as the operation to be performed on the business object.

The name of the business object is generated by the Adapter Connection wizard in accordance with the naming convention for the adapter.

The Siebel business objects are created with long names by default. To generate business objects with shorter names, select **Generate business objects with shorter names** on the Configure Objects screen of the Adapter Connection wizard.

For more information about business objects, see the following topics.

- "Naming conventions for business objects representing Siebel business services" on page 1404
- "Supported data operations (Siebel)"

Supported data operations (Siebel):

An operation is the action that an adapter can perform on the Siebel server during outbound processing. The name of the operation typically indicates the type of action that the adapter takes, such as create or update.

Table 195. Supported operations of business objects

Operation	Definition
Create	Creates the business component
Delete	Deletes the business component and its children
Exists	Checks for the existence of incoming business objects
Retrieve	Retrieves the values of the business component
Retrieve all	Retrieves multiple instances of the same business component

Table 195. Supported operations of business objects (continued)

Operation	Definition
Update	Updates the Siebel application with the incoming object

Naming conventions for business objects representing Siebel business services:

When the Adapter Connection wizard generates a business object, it provides a name for the business object based on the name of the object in the Siebel application that it uses to build the business object.

Naming conventions for business objects that represent Siebel business services and integration components

The naming conventions for business objects that represent Siebel business services are the same for both inbound and outbound processing. The names comprise the concatenation of several words, including prefix, business service name, integration object, and method name.

The following table describes the naming conventions that the Adapter Connection wizard uses to name business objects that represent Siebel business services and integration components.

Table 196. Business object naming conventions for Siebel business services and integration components

Element	Naming convention
Name of the business graph	*<Top Level business object Name>* +"BG" A "Prefix" is used only for top-level business objects that are generated against business service methods.
Name of the top-level business object	*<Prefix><Business Service Name><Method Name><Names of all the integration objects selected for the Input and InputOutput complex type arguments>* • If no Input or InputOutput arguments exist, the names of all the output arguments are: *<Prefix><Business Service Name><Method Name><Names of all the integration objects selected for the output complex type arguments>* • If the method contains no complex arguments in the method, the naming convention is: *<Prefix><Business Service Name><Method Name>*
Name of the inbound object that is generated against integration components	'IO' + *<Name of Integration Object>* + 'IC' + *<Name of integration component>* + 'BG' The top-level business graph has the suffix BG added to the business object name, as shown in this example: `IOAccountInterfaceICAccountBG`
Name of the outbound object that is generated against integration components	'IO' + *<Name of Integration Object>* + 'IC' + *<Name of integration component>* The name of an account interface integration object with the integration component account, as shown in the following example: `IOAccountInterfaceICAccount`

Optional: Shorter naming conventions for business objects that are generated against Siebel business services and integration components

The naming conventions for business objects that are generated against Siebel business services and integration components are valid if the optional 'Generate business objects with shorter names' property is specified on the configuration objects pane in the Adapter Connection wizard.

If this optional property is used, you should set the 'Folder' property with a unique value to avoid overwriting existing xsds that were previously generated. For example, if you select 'EAI Siebel Adapter', and click **Query** in two different Adapter Connection wizard runs for the integration objects, 'Account (PRM ANI)' and 'ATP Check Interface', the top-level object is named `EAISiebelAdapter.xsd`.

The name comprises the concatenation of several words, including prefix, business service name, and integration component name.

The following table describes the naming conventions that the Adapter Connection wizard uses to name business objects that are generated against Siebel business services and integration components.

Table 197. Shorter business object naming conventions for business objects that are generated against Siebel business services and integration components

Element	Naming convention
Name of the inbound and outbound child business objects that are generated against integration components	*<Prefix>+<Name of the Siebel Integration Component>* The Siebel business object and integration component names are stripped of all non-alphanumeric characters before being added to the business object name. If the resulting names are not unique, a counter is added to the end of the names.
Name of the inbound top-level business object that is generated against business services and integration components	*<Prefix>+<Name of the Siebel Integration Component>* +BG (with business graph specified) and *<Prefix>+<Name of the Siebel Integration Component>* (without business graph specified) The Siebel business object and integration component names are stripped of all non-alphanumeric characters before being added to the business object name. If the resulting names are not unique, a counter is added to the end of the names.
Name of the outbound top-level business object that is generated against business services and integration components	*<Prefix>+<Name of the Siebel Business Service>* +BG (with business graph specified) and *<Prefix>+<Name of the Siebel Business Service>* (without business graph specified) The Siebel business object and integration component names are stripped of all non-alphanumeric characters before being added to the business object name. If the resulting names are not unique, a counter is added to the end of the names.

Naming conventions for business objects that represent Siebel business objects

The naming conventions for business objects that represent Siebel business objects are the same for both inbound and outbound processing. The name comprises the concatenation of several words, including prefix, business object name, and business component name.

The following table describes the naming conventions that are used by the Adapter Connection wizard to name business objects that represent Siebel business objects.

Table 198. Business object naming conventions for Siebel business objects

Element	Naming convention
Name of the business object	<*Prefix*>+<BO>+<*Business Object Name*>+<BC>+<*Business Component Name*>. The Siebel business object and component names are stripped of all non-alphanumeric characters before being added to the business object name. If the resulting names are not unique, a counter is added to the end of the names. For example, if two business objects have the name, `SiebelBOAccountBCBusinessAddress`, a counter is added as a suffix to make them unique, as shown in this example: `SiebelBOAccountBCAddress1 and SiebelBOAccountBCAddress2`
Name of the container business object that is generated for the Exists operation	<SiebelExistsResult> A business graph is not generated for the "SiebelExistsResult" business object.
Name of the container business object that is generated for the Retrieve All operation	<*Prefix*>+BO+<*Business Object Name*>+<BC>+<*Business Component Name*>+Container
Name of the top-level business object	<*Prefix*>+<BO>+<*Business Object Name*>+<BC>+<*Business Component Name*>+BG The top-level business object does have a business graph generated.

Optional: Shorter naming conventions for business objects that are generated against Siebel business components

The naming conventions for business objects that are generated against Siebel business components are valid if the optional 'Generate business objects with shorter names' property is specified on the configuration objects pane of the Adapter Connection wizard.

If this optional property is used, set the 'Folder' property with a unique value to avoid overwriting existing xsds that were previously generated. For example, **Siebel business object** → **Siebel business component** combination of **Account-ESP** → **Account** and Account (as the top-level object) is named `Account.xsd`.

The name comprises the concatenation of several words, including prefix and business component name.

The following table describes the naming conventions that the Adapter Connection wizard uses to name business objects that are generated against Siebel business components.

Table 199. Shorter business object naming conventions for business objects that are generated against Siebel business components

Element	Naming convention
Name of the top-level business object that is generated against business components	<*Prefix*>+<*Name of the Siebel Business Component*> +BG (with business graph specified) and <*Prefix*>+<*Name of the Siebel Business Component*> (without business graph specified) The Siebel business object and integration component names are stripped of all non-alphanumeric characters before being added to the business object name. If the resulting names are not unique, a counter is added to the end of the names.

Configuration properties for the WebSphere Adapter for Siebel Business Applications

WebSphere Adapter for Siebel Business Applications has several categories of configuration properties, which you set with the Adapter Connection wizard when you generate or create objects and services.

You can change the resource adapter, managed connection factory, and activation specification properties in WebSphere Message Broker.

For more information, see the following topics.

- "Siebel connection properties for the Adapter Connection wizard"
- "Resource adapter properties (Siebel)" on page 1414
- "Managed connection factory properties (Siebel)" on page 1415
- "Activation specification properties (Siebel)" on page 1418

Siebel connection properties for the Adapter Connection wizard:

Set Adapter Connection wizard properties to establish a connection between the wizard, a tool that is used to create business objects, and the Siebel server. The properties that you configure in the Adapter Connection wizard specify such things as connection configuration, and logging and tracing options.

After you have established a connection between the Adapter Connection wizard and the Siebel server, the Adapter Connection wizard is able to access the metadata that it needs from the Siebel server to create business objects.

Some of the properties that you set in the Adapter Connection wizard are used as the initial value for resource adapter, managed connection factory, and activation specification properties that you can specify at a later time in the wizard.

The following table describes the Adapter Connection wizard properties and their purpose. A complete description of each property is provided in the sections that follow the table.

If you set any of these connection properties using bidirectional script, you must set values that identify the format of the bidirectional script that is entered for that property.

Table 200. Adapter Connection wizard properties

Property name in the wizard	Description
Adapter style	The service type that is associated with the adapter module
Connection URL	The connection URL that you need to connect to the Siebel application
Delimiter for keys in the event store	Specifies that the delimiter that is used between two name value pairs contains the object key name and value
Folder	The location of the generated business object
Generate business objects with shorter names	Ensures that the adapter generates shorter business object names, which are based on the Siebel integration components, business services, and business components rather than the concatenation of several words (which is the default)
Language code	The language code that is used to log on to the Siebel server
Method name	The name of the business service method to be implemented
Password	The password for the corresponding user name
Prefix for business object names	The prefix for the business object name
Siebel repository name	The name of the Siebel repository from which the objects are to be discovered
Siebel server view mode	Specifies the Siebel server mode and controls the kind of data to be retrieved and what actions can be performed
Type of Siebel objects to discover	The type of Siebel objects (business objects or business services) that need to be discovered and listed
Use resonate support for load balancing on Siebel server	Specifies that if resonate support is installed on the Siebel server, and the value is set to `True`, the adapter takes advantage of the load balancing feature to connect to the Siebel server more efficiently
User name	The user name that is used to log into the Siebel application

Adapter style (AdapterStyle)

This property specifies the service type that is associated with the adapter module.

Table 201. Service type details

Required	Yes
Default	Outbound
Property type	List of values
Possible values	`Outbound` `Inbound`
Usage	Specifies the service type associated with the adapter module
Globalized	No
Bidi supported	No

Business object namespace (BusinessObjectNameSpace)

This property specifies that the namepsace value has been added as a prefix to the business object name to keep the business object schemas separated.

Table 202. Business object namespace details

Required	Yes
Default	`http://www.ibm.com/xmlns/prod/wbi/j2ca/siebel`
Property type	String
Usage	The namespace value is added as a prefix to the business object name to keep the business object schemas separated.
Example	`http://www.ibm.com/xmlns/prod/wbi/j2ca/siebel/IBMSiebelAccountInsertAccount`
Globalized	No
Bidi supported	No

Connection URL (ConnectionURL)

This property specifies the connection URL that is needed to connect to the Siebel application.

Table 203. Connection URL details

Required	Yes
Default	No default value
Property type	String
Usage	The connection URLs for all versions of Siebel follow this format: `Protocol:// machinename:port/enterprisename/object manager/server name.` . The default port number is 2320. For Siebel version 7.5x and earlier versions, the port number (2320) and server name are specified. For Siebel version 7.8, the port and server name are not specified. If you do not select the default port, you can specify another port number (for example, 2321).
Examples	The following sample connection URLs are for different versions of Siebel: • For Siebel 7.5: `siebel://<IP_address>:2320/siebel/SSEObjMgr_ENU/sebldevl`. • For Siebel 7.8: `siebel://<IP_address>/Sieb78/SSEObjMgr_enu`. • For Siebel 8: `siebel://<IP_address>:2321/SBA_80/SSEObjMgr_enu`.
Globalized	Yes
Bidi supported	Yes

Delimiter for keys in the event store (DelimiterForKeysInTheEventStore)

Table 204. Delimiter for keys in the event store details

Required	Yes
Default	;
Property type	String
Usage	This is the delimiter that is used between two name value pairs that contain the object key name and value.
Examples	You can change the default value for this property. However, if you remove the default value and do not set it again, the default value (;) is used. If the event table key field has values, such as `AccountId=1-314:Id=1-325`, the event delimiter is the colon (:) . The object key names are `AccountId` and `Id`. The values are `1-314` and `1-325`.
Globalized	Yes

Table 204. Delimiter for keys in the event store details (continued)

Bidi supported	Yes

Folder (Folder)

This property specifies the location of the generated business objects.

Table 205. Folder details

Required	No
Default	No default value
Property type	String
Usage	The generated business objects are copied into this folder.
Example	`inboundartifacts` and `outboundartifacts`
Globalized	No
Bidi supported	No

Generate business objects with shorter names (GenerateBusinessObjectsWithShorterNames)

This property ensures that the adapter generates shorter business object names, which are based on the Siebel integration components, business services, and business components rather than the concatenation of several words (which is the default).

Table 206. Generate business objects with shorter names details

Required	No
Default	No default value
Property type	Boolean
Usage	This property ensures that the adapter generates shorter business object names. The shorter business object names are based on the Siebel integration components, business services, and business components. The prefix is also attached to the shorter names. The adapter removes special characters from the shorter business object names. Alphanumeric characters (a-z, A-Z, and 1-9) are supported, and a counter (1-9) is added to the end of business object names to avoid duplication of names.
Example	If 'Account' is the name of the Siebel business component, and 'Siebel' is the prefix, the shorter name is 'Siebel_Account'.
Globalized	No
Bidi supported	No

Language code (LanguageCode)

This property specifies the language code that is used to log on to the Siebel server.

Table 207. Language code details

Required	Yes
Default	`ENU`
Property type	String

Table 207. Language code details (continued)

Usage	If the system locale is English, the value for this property is `ENU` (English). This is used to log on to the Siebel server.
Globalized	No
Bidi supported	No

Method name (MethodName)

This property specifies the name of the business service method to be implemented.

Table 208. Method name details

Required	Yes
Default	`Query`
Property type	String
Usage	The default is `Query`.
Example	`Query`, `QueryByExample`, `QueryById`, and so on.
Globalized	Yes
Bidi supported	Yes

Password (Password)

This property specifies the password for the corresponding user name.

Table 209. Password details

Required	Yes
Default	No default value
Property type	String
Usage	If a J2C Authentication Alias is used, a password is not required.
Example	`1-XYZ`
Globalized	Yes
Bidi supported	Yes

Prefix for business object names (PrefixForBusinessObjectNames)

This property specifies the prefix for the business object name.

Table 210. Prefix details

Required	No
Default	No default value
Property type	String
Usage	The prefix string is attached to the front of the business object name that was generated.

Table 210. Prefix details (continued)

Example	For example, you use the prefix, `IBM`, and generate a business object for the EAI Siebel Adapter and the Insert method. Then you choose the Account Interface and Business Address Interface integration object against an Input and InputOutput method argument. The corresponding business object that is generated is: `IBMEAISiebelAdapterInsertAccountInterfacBusinessAddressInterface` .
Globalized	Yes
Bidi supported	Yes

Siebel business object name for event store (SiebelBusinessObjectNameForEventStore)

This property specifies the name of the business object in the event store where events are stored for inbound processing.

Table 211. Siebel business object name for event store details

Required	Yes
Default	`IBM_EVENT`
Property type	String
Usage	After you click **Advanced** on the connection properties pane of the Adapter Connection wizard, this property displays on the **Event configuration** tab. The two values that are listed are `IBM_EVENT` and `IBM2`. If you create a custom event component name, you can specify the value for it in the list box.
Globalized	Yes
Bidi supported	No

Siebel repository name (SiebelRepositoryName)

This property specifies the name of the Siebel repository from which the objects are discovered.

Table 212. Siebel repository name details

Required	Yes
Default	`Siebel repository`
Property type	String
Usage	This default value is `Siebel Repository`. Although this is a required field, it is optional on the Adapter Connection wizard. You can edit this value to point to other repositories if needed.
Globalized	No
Bidi supported	No

Siebel server view mode (SiebelServerViewMode)

This property specifies the Siebel server view mode and controls the data that can be retrieved and what actions can be performed on it.

Table 213. Siebel server view mode details

Required	Yes

Table 213. Siebel server view mode details (continued)

Default	3
Property type	Integer
Usage	This property displays when you click **Advanced** on the connection properties pane of the Adapter Connection wizard. This mode, when set to "Type of Siebel objects to discover" applies only to `Siebel business objects`, not to `Siebel business services`. The values that are supported by Siebel are 1 to 9.
Globalized	No

Type of Siebel objects to discover (TypeOfSiebelObjectsToDiscover)

This property specifies the type of Siebel object that needs to be discovered and listed.

Table 214. Type of Siebel objects to discover details

Required	Yes
Possible values	`Siebel business objects` and `Siebel business services`
Default	`Siebel business objects`
Property type	String
Usage	Although the default is `Siebel business objects`, you can select `Siebel business services`. Based on your selection, the Adapter Connection wizard retrieves either the business objects or the business services.
Globalized	No
Bidi supported	No

Use resonate support for load balancing on Siebel server (UseResonateSupportForLoadBalancingOnSiebelServer)

This property indicates whether the Siebel server uses resonate support.

Table 215. Use resonate support for load balancing on Siebel server details

Required	No
Possible values	`True` `False`
Default	`True`
Property type	Boolean
Usage	This property displays after you click **Advanced** on the connection properties pane of the Adapter Connection wizard. If you select the check box, the property is set to `True` and the adapter takes advantage of the load balancing feature to connect to the Siebel server more efficiently. If you clear the check box, the property is set to `False`.
Globalized	No

User name (UserName)

This property specifies the user name that is used to log into the Siebel application.

Table 216. User name details

Required	Yes
Default	No default value

Table 216. User name details (continued)

Property type	String
Usage	If a J2C Authentication Alias is used, a user name is not required.
Globalized	Yes
Bidi supported	Yes

Resource adapter properties (Siebel):

The resource adapter properties control the general operation of the adapter. You set the resource adapter properties using the Adapter Connection wizard when you configure the adapter.

The following table lists and describes the resource adapter properties. A more detailed description of each property is provided in the sections that follow the table.

Table 217. Resource adapter properties

Property name	Description
Adapter ID property	The adapter instance for CEI and PMI events with respect to logging and tracing.
Event delimiter	Specifies whether the delimiter that is used between two name value pairs contains the object key name and value
Resonate support	Specifies that if resonate support is installed on the Siebel server, and the value is set to `True`, the adapter takes advantage of the load balancing feature to connect to the Siebel server more efficiently
Siebel server view mode	Specifies the Siebel view mode and controls the kind of data to be retrieved and what actions can be performed

Adapter ID to use for logging and tracing (AdapterID)

This property identifies a specific deployment, or instance, of the adapter.

Table 218. Adapter ID to use for logging and tracing details

Required	Yes
Default	CWYMY_Adapter Without local transaction support: CWYAP_SAPAdapter With local transaction support: CWYAP_SAPAdapter_Tx
Property type	String
Usage	Use this property to identify the adapter instance for PMI events. If you are deploying multiple instances of an adapter, set this property to a unique value for each adapter instance. For inbound processing, this property is retrieved from the resource adapter properties. For outbound processing, it is retrieved from the managed connection factory properties.
Globalized	Yes
Bidi supported	No

Event delimiter (EventDelimiter)

This property indicates that the delimiter that is used between two name value pairs contains the object key name and value.

Table 219. Event delimiter details

Required	Yes
Default	;
Property type	String
Usage	If multiple value pairs are set against the object key in the event component, they are used for the delimiter.
Globalized	No

Resonate support (ResonateSupport)

This property indicates whether the Siebel server uses resonate support.

Table 220. Resonate support details

Required	No
Possible values	`True` `False`
Default	`True`
Property type	Boolean
Usage	If you select the check box, the value for Resonate Support is set to `True`, and the adapter takes advantage of the load balancing feature to connect to the Siebel server more efficiently. If you clear the check box, the value for Resonate Support is set to `False`.
Globalized	No

Siebel server view mode (SiebelServerViewMode)

This property specifies the Siebel view mode and controls the data that can be retrieved and what actions can be performed on it.

Table 221. View mode details

Required	Yes
Default	3
Property type	Integer
Usage	The View mode property applies only to Siebel business objects and not to Siebel business services.
Globalized	No

Managed connection factory properties (Siebel):

The adapter uses managed connection factory properties at run time to create an outbound connection instance with the Siebel application.

Use the Adapter Connection wizard to set the managed connection factory properties.

The following table lists the managed connection factory properties for inbound communication. A complete description of each property is provided in the sections that follow the table.

Table 222. Managed connection factory properties

Property name	Description
Connection URL	The connection URL that is needed to connect to the Siebel application
Language code	The language code that is used to log on to the Siebel server
Password	The password for the corresponding user name
Prefix	The prefix for the business object name
Resonate support	Specifies that if resonate support is installed on the Siebel server, and the value is set to `True`, the adapter takes advantage of the load balancing feature to connect to the Siebel server more efficiently
User name	The user name that is used to log into the Siebel application
View mode	Specifies the Siebel view mode and controls the data that can be retrieved and what actions can be performed on it

Connection URL (ConnectionURL)

This property specifies the connection URL that is needed to connect to the Siebel application.

Table 223. Connection URL details

Required	Yes
Default	No default value
Property type	String
Usage	`Protocol://machinename:port/enterprisename/object manager/server name` For Siebel 7.0.5 to 7.5x : `siebel://<IP ADDRESS>/siebel/SSEObjMgr_ENU/sebldev1` For Siebel 7.8 : `siebel://<IP ADDRESS>:2321/Sieb78/SSEObjMgr_enu` For Siebel 8 : `siebel://<IP ADDRESS>:2321/SBA_80/SSEObjMgr_enu` The default port number is 2320. In the examples above (for Siebel versions 7.8 and 8) another port (2321) has been specified.
Globalized	Yes
Bidi supported	Yes

Language code (LanguageCode)

This property specifies the language code that is used to log on to the Siebel server.

Table 224. Language code details

Required	Yes
Possible values	None
Default	`ENU`

Table 224. Language code details (continued)

Property type	String
Usage	If the system locale is English, the value for this property is `ENU` (English). This value is used to log on to the Siebel server.
Globalized	No
Bidi supported	No

Password (Password)

This property specifies the password for the corresponding user name.

Table 225. Password details

Required	Yes
Default	No default value
Property type	String
Example	`sadmin`
Usage	This property displays after you click **Advanced** on the connection properties pane of the external service wizard. The password is saved in `.import` and `.export` files so that the adapter can connect to the Siebel application after it has been deployed. If a J2C Authentication Alias is used, a password is not required.
Globalized	Yes
Bidi supported	Yes

Prefix (Prefix)

This property specifies the prefix for the business object name.

Table 226. Prefix details

Required	No
Default	No default value
Property type	String
Usage	The prefix string is attached to the front of the business object name.
Example	If you use the prefix `IBM`, generate a business object for the EAI Siebel Adapter and the Insert method, and choose the integration object, Account (PRM ANI), the corresponding business object generated is: `IBMEAISiebelAdapterInsertAccountU40PRMANIU41` where U40 and U41 are the unicode value replacements of `(and)`.
Globalized	Yes
Bidi supported	Yes

Resonate support (ResonateSupport)

This property indicates whether the Siebel server uses resonate support.

Table 227. Resonate support details

Required	No

Table 227. Resonate support details (continued)

Possible values	`True` `False`
Default	`True`
Property type	Boolean
Usage	If you select the check box, the property is set to `True`, and the adapter takes advantage of the load balancing feature to connect to the Siebel server more efficiently. If you clear the check box, the property is set to `False`.
Globalized	No

User name (UserName)

This property specifies the user name that is used to log into the Siebel application.

Table 228. User name details

Required	Yes
Possible values	None
Default	No default value
Property type	String
Usage	This property displays after you click **Advanced** on the connection properties pane of the Adapter Connection wizard. The user name is saved in `.import` and `.export` files so that the adapter can connect to the Siebel application after it has been deployed. If a J2C Authentication Alias is used, a password is not required.
Globalized	Yes
Bidi supported	Yes

View mode (ViewMode)

This property specifies the Siebel view mode and controls the data that can be retrieved and what actions can be performed on it.

Table 229. View mode details

Required	Yes
Default	Although the adapter supports values 1 to 9, the default value is 3.
Property type	Integer
Usage	The View mode property applies only to Siebel business objects and not to Siebel business services. When this property is used for Siebel business objects, the default value is 3.
Example	The adapter supports values 1 to 9. For example, 1 is Manager View, 2 is Personal View, and 3 is All View.
Globalized	No

Activation specification properties (Siebel):

Activation specification properties hold the configuration information for inbound event processing of a message endpoint. Use the Adapter Connection wizard to set activation specification properties.

Activation specification properties are used during endpoint activation to notify the adapter of eligible event listeners. During inbound processing, the adapter uses these event listeners to receive events before it forwards them to the endpoint.

The following table lists the activation specification properties for inbound processing that you set using the Adapter Connection wizard. A more detailed description of each property is provided in the sections that follow the table.

Table 230. Activation specification properties

Property name	Description
Connection URL	The connection URL that is needed to connect to the Siebel application
Delivery type	Determines the order in which events are delivered by the adapter to the export component
Do not process events that have a timestamp in the future	Specifies whether the adapter filters out future events by comparing the timestamp on each event with the system time
Ensure once-only event delivery	Specifies whether the adapter provides assured once-only delivery of events
Event component name	The name of the Siebel component for the event table
Event types to process	A delimited list of event types that indicates to the adapter which events it should deliver
Interval between polling periods	The length of time that the adapter waits between polling periods
Language code	The language code that is used to log on to the Siebel server
Maximum connections	The maximum number of connections that the adapter can use for inbound event delivery
Maximum events in polling period	The number of events that the adapter delivers to the export component during each poll period
Minimum connections	The minimum number of connections that the adapter can use for inbound event delivery
Number of times to retry the system connection	The number of times that the adapter tries to reestablish an inbound connection after an error occurs.
Password	The password for the corresponding user name
Retry interval	The length of time that the adapter waits between attempts to establish a new connection after an error occurs during inbound operations
Stop the adapter when an error is encountered while polling	Specifies whether the adapter stops polling for events when it encounters an error during polling.
User name	The user name that is used to log on to the Siebel application

Connection URL (ConnectionURL)

This property specifies the connection URL that is needed to connect to the Siebel application.

Table 231. Connection URL details

Required	Yes
Default	No default value

Table 231. Connection URL details (continued)

Property type	String
Usage	`Protocol://machinename:port/enterprisename/object manager/server name` For Siebel 7.0.5 to 7.5x : `siebel://<IP ADDRESS>/siebel/SSEObjMgr_ENU/sebldev1` For Siebel 7.8 : `siebel://<IP ADDRESS>:2321/Sieb78/SSEObjMgr_enu` For Siebel 8 : `siebel://<IP ADDRESS>:2321/SBA_80/SSEObjMgr_enu` The default port number is 2320. In the examples above (for Siebel versions 7.8 and 8), another port (2321) has been specified.
Globalized	Yes
Bidi supported	Yes

Delivery type (DeliveryType)

This property specifies the order in which events are delivered by the adapter to the export component.

Table 232. Delivery type details

Required	No
Possible values	`ORDERED` `UNORDERED`
Default	`ORDERED`
Property type	String
Usage	The following values are supported: • `ORDERED`: The adapter delivers events to the export component one at a time. • `UNORDERED`: The adapter delivers all events to the export component at once.
Globalized	No
Bidi supported	No

Do not process events that have a timestamp in the future (FilterFutureEvents)

This property specifies whether the adapter filters out future events by comparing the timestamp on each event with the system time.

Table 233. Do not process events that have a timestamp in the future details

Required	Yes
Possible values	`True` `False`
Default	`False`
Property type	Boolean
Usage	If set to `True`, the adapter compares the time of each event to the system time. If the event time is later than the system time, the event is not delivered. If set to `False`, the adapter delivers all events.
Globalized	No
Bidi supported	No

Ensure once-only event delivery (AssuredOnceDelivery)

This property specifies whether to provide ensured once-only event delivery for inbound events.

Table 234. Ensure once-only event delivery details

Required	Yes
Possible values	`True` `False`
Default	`True`
Property type	Boolean
Usage	When this property is set to `True`, the adapter provides assured once-only event delivery so that each event is delivered only once. A value of `False` does not provide assured once event delivery, but provides better performance. When this property is set to `True`, the adapter attempts to store transaction (XID) information in the event store. If it is set to `False`, the adapter does not attempt to store the information. This property is used only if the export component is transactional. If it is not, no transaction can be used, regardless of the value of this property.
Globalized	No
Bidi supported	No

Event component name (EventComponentName)

This property specifies the name of the event store where events are stored for inbound processing.

Table 235. Event component name details

Required	Yes
Default	`IBM2` (for Siebel version 7.x) and `IBM Event` (for Siebel version 8)
Property type	String
Usage	The default value is `IBM2` for Siebel version 7.x, and `IBM Event` for Siebel version 8. If you select one of these default values to configure the event business component on the Siebel server, it is the name of the Siebel event business component that was created. You can also select a value from the list of values provided by the adapter. You can edit the list of values. If you create your own Siebel event business component, you can edit the list to include the name of that event business component.
Globalized	Yes
Bidi supported	Yes

Event types to process (EventTypeFilter)

This property contains a delimited list of event types that indicates to the adapter which events it should deliver.

Table 236. Event types to process details

Required	No
Possible values	A comma-delimited (,) list of business object types
Default	`null`
Property type	String

Table 236. Event types to process details (continued)

Usage	Events are filtered by business object type. If the property is set, the adapter delivers only those events that are in the list. A value of `null` indicates that no filter will be applied and that all events will be delivered to the export component.
Example	To receive only events that relate to the Customer and Order business objects, specify this value: `Customer,Order`
Globalized	No
Bidi supported	No

Interval between polling periods (PollPeriod)

This property specifies the length of time that the adapter waits between polling periods.

Table 237. Interval between polling periods details

Required	Yes
Possible values	Integers greater than or equal to `0`.
Default	`2000`
Unit of measure	Milliseconds
Property type	Integer
Usage	The poll period is established at a fixed rate, which means that if running the poll cycle is delayed for any reason (for example, if a prior poll cycle takes longer than expected to complete) the next poll cycle occurs immediately to make up for the lost time that was caused by the delay.
Globalized	No
Bidi supported	No

Language code (LanguageCode)

This property specifies the language code that is used to log on to the Siebel server.

Table 238. Language code details

Required	Yes
Default	`ENU`
Property type	String
Usage	If the system locale is English, the value for this property is `ENU` (English), which is used to log on to the Siebel server.
Globalized	No
Bidi supported	No

Maximum connections (MaximumConnections)

This property specifies the maximum number of connections that the adapter can use for inbound event delivery.

Table 239. Maximum connections details

Required	No
Default	1
Property type	Integer
Usage	Only positive values are valid. The adapter considers any positive entry less than 1 to be equal to 1. Typing a negative value or 0 for this property might result in runtime errors.
Globalized	No
Bidi supported	No

Maximum events in polling period (PollQuantity)

This property specifies the number of events that the adapter delivers to the export component during each poll period.

Table 240. Maximum events in polling period details

Required	Yes
Default	10
Property type	Integer
Usage	The value must be greater than 0. If this value is increased, more events are processed per polling period, and the adapter might perform less efficiently. If this value is decreased, fewer events are processed per polling period, and the adapter's performance might improve slightly.
Globalized	No
Bidi supported	No

Minimum connections (MinimumConnections)

This property specifies the minimum number of connections that the adapter can use for inbound event delivery.

Table 241. Minimum connections details

Required	No
Default	1
Property type	Integer
Usage	Only positive values are valid. Any value less than 1 is treated as 1 by the adapter. Typing a negative value or 0 for this property might result in runtime errors.
Globalized	No
Bidi supported	No

Number of times to retry the system connection (RetryLimit)

This property specifies the number of times that the adapter tries to reestablish an inbound connection.

Table 242. Number of times to retry the system connection details

Required	No
Possible values	Positive integers
Default	0

Table 242. Number of times to retry the system connection details (continued)

Property type	Integer
Usage	Only positive values are valid. When the adapter encounters an error that is related to the inbound connection, this property specifies the number of times that the adapter tries to restart the connection. A value of `0` indicates an infinite number of retries.
Globalized	Yes
Bidi supported	No

Password (Password)

This property specifies the password for the corresponding user name.

Table 243. Password details

Required	Yes
Default	No default value
Property type	String
Usage	This property displays after you click **Advanced** on the connection properties pane of the Adapter Connection wizard. The password is saved in `.import` and `.export` files so that the adapter can connect to the Siebel application after it has been deployed. If a J2C Authentication Alias is used, a password is not required.
Example	`sadmin`
Globalized	Yes
Bidi supported	Yes

Retry interval if connection fails (RetryInterval)

When the adapter encounters an error that is related to the inbound connection, this property specifies the length of time that the adapter waits before trying to establish a new connection.

Table 244. Retry interval details

Required	Yes
Default	`2000`
Unit of measure	Milliseconds
Property type	Integer
Usage	Only positive values are valid. When the adapter encounters an error that is related to the inbound connection, this property specifies the length of time that the adapter waits before trying to establish a new connection.
Globalized	Yes
Bidi supported	No

Stop the adapter when an error is encountered while polling (StopPollingOnError)

This property specifies whether the adapter stops polling for events when it encounters an error during polling.

Table 245. Stop the adapter when an error is encountered while polling details

Required	No
Possible values	`True` `False`
Default	`False`
Property type	Boolean
Usage	If this property is set to `True`, the adapter stops polling when it encounters an error. If this property is set to `False`, the adapter logs an exception when it encounters an error during polling and continues polling.
Globalized	No
Bidi supported	No

User name (UserName)

This property specifies the user name that is used to log into the Siebel application.

Table 246. User name details

Required	Yes
Default	No default value
Property type	String
Usage	This property displays after you click **Advanced** on the connection properties pane of the Adapter Connection wizard. The user name is saved in `.import` and `.export` files so that the adapter can connect to the Siebel application after it has been deployed. If a J2C Authentication Alias is used, a password is not required.
Globalized	Yes
Bidi supported	Yes

WebSphere Adapter for PeopleSoft properties

Reference information to refer to when you connect to a PeopleSoft application.

- "Business-object information (PeopleSoft)"
- "Supported operations (PeopleSoft)" on page 1426
- "PeopleCode for a custom event project" on page 1427
- "Configuration properties for the WebSphere Adapter for PeopleSoft Enterprise" on page 1431
- "PeopleSoft connection properties for the Adapter Connection wizard" on page 1431
- "Resource adapter properties (PeopleSoft)" on page 1434
- "Managed connection factory properties (PeopleSoft)" on page 1436
- "Activation specification properties (PeopleSoft)" on page 1439
- "Interaction specification properties (PeopleSoft)" on page 1444

Business-object information (PeopleSoft)

Information about the content of a business object is located primarily inside the business-object definition file, a file that is generated by the Adapter Connection wizard when it creates business objects.

The business-object definition file contains application-specific information (ASI), which the adapter uses to perform operations, such as creating or updating. You can also find information about the content of a business object in the name of the business object. Although business-object names have no semantic value, they often contain clues about what the business object contains and what operation the adapter can perform on the PeopleSoft Enterprise server. For example, a business object named UpdateAddress suggests that the adapter can update an address in the PeopleSoft Enterprise server.

For more information, see "Supported operations (PeopleSoft)."

Supported operations (PeopleSoft):

An operation is the action that an adapter can perform on the PeopleSoft Enterprise server during outbound processing. The name of the operation typically indicates the type of action that the adapter takes, such as create or update.

The following table defines the operations that the adapter supports.

Table 247. Supported operations of business objects

Operation	Definition
Create	The adapter accesses the PeopleSoft component and retrieves values from the attributes that have the primary key application-specific information set. The adapter then opens the corresponding component interface by using the value that is provided for the ObjectName application-specific information. It sets the attribute values on the corresponding Create Keys in the component interface. An empty Component Interface is created, and the adapter maps all the business-object data to the created component interface. When mapping the data, the adapter sends all data for simple attributes in the hierarchy, and it creates items that match each of the child objects in the hierarchy, including effective-dated and effective-sequenced child records.
Retrieve	The adapter accesses the PeopleSoft component and retrieves values from the attributes that have the primary key application-specific information set. The adapter then instantiates the corresponding component interface by using the value that is provided for the ObjectName application-specific information. It sets the attribute values on the corresponding Get Keys in the component interface. The adapter then maps the component data onto the business-object hierarchy. Child objects are included in the data mapping.
RetrieveAll	This operation performs in the same way as the Retrieve operation, except that it allows retrieval of multiple instances of the same PeopleSoft component.
Update	The adapter retrieves an object from PeopleSoft and compares it to the target business object. When the comparison reveals extra child objects in PeopleSoft, the adapter deletes children. When the comparison reveals missing children in PeopleSoft, the adapter creates children. When the comparison reveals child objects that have been updated in PeopleSoft, the adapter updates them.
Exists	The adapter processes an exist operation in the same way that it processes a retrieve operation, except that it does not populate the business object with retrieved data; it checks for the existence of an object in PeopleSoft.
Delete	Based on the values that are set for the application-specific metadata elements StatusColumnName and StatusValue, the adapter updates a business object to inactive. A delete operation can be performed only on a top level object. PeopleSoft does not allow an object to be physically deleted, so the inactive object remains in the PeopleSoft database.
Apply Changes	This operation updates the PeopleSoft component based on the operation that was performed on it. The supported operations are create, update, and delete.

PeopleCode for a custom event project

Two PeopleCode functions are required to support inbound processing. If you create a custom event project in PeopleTools for inbound support, add the PeopleCode functions to the project.

The following PeopleCode contains the IBMPublishEvent and IBMPublishFutureDatedEvent functions that are used to publish events to the event table. Calls to these functions are made from the SavePostChange PeopleCode function in the PeopleSoft component of interest.

```
/* IBM event notification */
Component string &KEYSTRING;
Component string &KEYNAME;
Component array of string &KEYARRAY;
Component string &KEYDELIM;
Component string &IBMVERB;
Local Record &IBMREC;

Function IBMPublishFutureDatedEvent(&BO, &KEYS, &EFFDATE)
; /* == create a new record object for cw_event_tbl == */
   &IBMREC = CreateRecord(Record.IBM_EVENT_TBL);
 /* =============== KEYS ================ */
/* composing keys and values in name value format */
   &KEYSTRING = "";
   &KEYDELIM = ":";
   &KEYARRAY = Split(&KEYS, &KEYDELIM);
   &LEN = &KEYARRAY.Len;
   For &I = 1 To &LEN;
/* get keys and values */
/* get rid of record name */
      &POS1 = Find(".", &KEYARRAY [&I]);
      &L1 = Len(&KEYARRAY [&I]);
      &POS2 = &L1 - &POS1;
      &KEYNAME = Right(&KEYARRAY [&I], &POS2);
/****The code below will remove special characters and****/
/****adjust the characters' case to ensure it is same as the****/
/****attribute name in the business object definition***/
/****Start****/
      &lLen = Len(&KEYNAME);
      &sOrigString = &KEYNAME;
      &sNewString = "";
      &lCtr2 = 1;
      &isSpecialChar = "true";
      For &lCtr = 1 To &lLen;
         &sChar = Substring(&sOrigString, &lCtr, 1);
         If (&sChar = "A" Or
               &sChar = "a" Or
               &sChar = "B" Or
               &sChar = "b" Or
               &sChar = "C" Or
               &sChar = "c" Or
               &sChar = "D" Or
               &sChar = "d" Or
               &sChar = "E" Or
               &sChar = "e" Or
               &sChar = "F" Or
               &sChar = "f" Or
               &sChar = "G" Or
               &sChar = "g" Or
               &sChar = "H" Or
               &sChar = "h" Or
               &sChar = "I" Or
               &sChar = "i" Or
               &sChar = "J" Or
               &sChar = "j" Or
               &sChar = "K" Or
               &sChar = "k" Or
               &sChar = "L" Or
               &sChar = "l" Or
               &sChar = "M" Or
               &sChar = "m" Or
               &sChar = "N" Or
               &sChar = "n" Or
               &sChar = "O" Or
               &sChar = "o" Or
               &sChar = "P" Or
               &sChar = "p" Or
               &sChar = "Q" Or
               &sChar = "q" Or
               &sChar = "R" Or
               &sChar = "r" Or
               &sChar = "S" Or
               &sChar = "s" Or
               &sChar = "T" Or
               &sChar = "t" Or
               &sChar = "U" Or
```

```
                &sChar = "u" Or
                &sChar = "V" Or
                &sChar = "v" Or
                &sChar = "W" Or
                &sChar = "w" Or
                &sChar = "X" Or
                &sChar = "x" Or
                &sChar = "Y" Or
                &sChar = "y" Or
                &sChar = "Z" Or
                &sChar = "z" Or
                &sChar = "1" Or
                &sChar = "2" Or
                &sChar = "3" Or
                &sChar = "4" Or
                &sChar = "5" Or
                &sChar = "6" Or
                &sChar = "7" Or
                &sChar = "8" Or
                &sChar = "9" Or
                &sChar = "0") Then
             If (&isSpecialChar = "true") Then
                &sNewString = &sNewString | Upper(&sChar);
                &isSpecialChar = "false";
             Else
                &sNewString = &sNewString | Lower(&sChar);
             End-If;
          Else
             &isSpecialChar = "true";
          End-If;
       End-For;
       &KEYNAME = &sNewString;
/*********End*********/
       &KEYSTRING = &KEYSTRING | &KEYNAME | "=" | @&KEYARRAY [&I] | &KEYDELIM
   End-For;
   &KEYSTRING = RTrim(&KEYSTRING, ":");
   &IBMREC.IBM_OBJECT_KEYS.Value = &KEYSTRING;
/*============== VERB ========================*/
/*  verb determination uses variable &IBMVERB */
   Evaluate %Mode
   When = "A"
      &IBMVERB = "Create";
      Break;
   When = "U"
      &IBMVERB = "Update";
      Break;
   When = "L"
      &IBMVERB = "Update";
      Break;
   When = "C"
      &IBMVERB = "Update";
      Break;
   When-Other
      &IBMVERB = "Retrieve";
   End-Evaluate;
   &IBMREC.IBM_OBJECT_VERB.Value = &IBMVERB;
/*  =====================  EVENT_ID GEN   ================================== */
/*  create event_id */
   &NEWNUM = GetNextNumber(IBM_FETCH_ID.IBM_NEXT_EVENT_ID, 99999);
/*  only use newnum if no error generating next number  */
   If &NEWNUM > 0 Then
      &IBMREC.IBM_EVENT_ID.Value = &NEWNUM;
   Else
      &IBMREC.IBM_EVENT_ID.Value = %Datetime;
   End-If; /*Support for Future Effective Date - The adapter will poll such events when the date arrives*/
   If &EFFDATE > %Datetime Then
      &IBMREC.IBM_EVENT_DTTM.Value = &EFFDATE;
      &IBMREC.IBM_EVENT_STATUS.Value = "99";
   Else
      &IBMREC.IBM_EVENT_DTTM.Value = %Datetime;
      &IBMREC.IBM_EVENT_STATUS.Value = "0";
   End-If; /*================  INSERT EVENT INTO IBM_EVENT_TBL ============*/
/*  insert row into table using record object*/
   &IBMREC.IBM_OBJECT_NAME.Value = &BO;
   &IBMREC.Insert();
End-Function;

Function IBMPublishEvent(&BO, &KEYS);
   /* == create a new record object for cw_event_tbl == */
   &IBMREC = CreateRecord(Record.IBM_EVENT_TBL);


   /* =============== KEYS ================ */
   /* composing keys and values in name value format */
   &KEYSTRING = "";
   &KEYDELIM = ":";
   &KEYARRAY = Split(&KEYS, &KEYDELIM);
```

```
&LEN = &KEYARRAY.Len;

For &I = 1 To &LEN;
   /* get keys and values */
   /* get rid of record name */
   &POS1 = Find(".", &KEYARRAY [&I]);
   &L1 = Len(&KEYARRAY [&I]);
   &POS2 = &L1 - &POS1;
   &KEYNAME = Right(&KEYARRAY [&I], &POS2);

   /****The code below will remove special characters and
   /****adjust the characters' case to ensure it is same as the
   /****attribute name in the business object definition***/
   /****Start****/
   &lLen = Len(&KEYNAME);
   &sOrigString = &KEYNAME;
   &sNewString = "";
   &lCtr2 = 1;
   &isSpecialChar = "true";
   For &lCtr = 1 To &lLen;
      &sChar = Substring(&sOrigString, &lCtr, 1);
      If (&sChar = "A" Or
            &sChar = "a" Or
            &sChar = "B" Or
            &sChar = "b" Or
            &sChar = "C" Or
            &sChar = "c" Or
            &sChar = "D" Or
            &sChar = "d" Or
            &sChar = "E" Or
            &sChar = "e" Or
            &sChar = "F" Or
            &sChar = "f" Or
            &sChar = "G" Or
            &sChar = "g" Or
            &sChar = "H" Or
            &sChar = "h" Or
            &sChar = "I" Or
            &sChar = "i" Or
            &sChar = "J" Or
            &sChar = "j" Or
            &sChar = "K" Or
            &sChar = "k" Or
            &sChar = "L" Or
            &sChar = "l" Or
            &sChar = "M" Or
            &sChar = "m" Or
            &sChar = "N" Or
            &sChar = "n" Or
            &sChar = "O" Or
            &sChar = "o" Or
            &sChar = "P" Or
            &sChar = "p" Or
            &sChar = "Q" Or
            &sChar = "q" Or
            &sChar = "R" Or
            &sChar = "r" Or
            &sChar = "S" Or
            &sChar = "s" Or
            &sChar = "T" Or
            &sChar = "t" Or
            &sChar = "U" Or
            &sChar = "u" Or
            &sChar = "V" Or
```

```
            &sChar = "v" Or
            &sChar = "W" Or
            &sChar = "w" Or
            &sChar = "X" Or
            &sChar = "x" Or
            &sChar = "Y" Or
            &sChar = "y" Or
            &sChar = "Z" Or
            &sChar = "z" Or
            &sChar = "1" Or
            &sChar = "2" Or
            &sChar = "3" Or
            &sChar = "4" Or
            &sChar = "5" Or
            &sChar = "6" Or
            &sChar = "7" Or
            &sChar = "8" Or
            &sChar = "9" Or
            &sChar = "0") Then
         If (&isSpecialChar = "true") Then
            &sNewString = &sNewString | Upper(&sChar);
            &isSpecialChar = "false";
         Else
            &sNewString = &sNewString | Lower(&sChar);
         End-If;
      Else
         &isSpecialChar = "true";
      End-If;
   End-For;
   &KEYNAME = &sNewString;

   /*********End*********/
   &KEYSTRING = &KEYSTRING | &KEYNAME | "=" | @&KEYARRAY [&I] | &KEYDELIM
End-For;
&KEYSTRING = RTrim(&KEYSTRING, ":");

&IBMREC.IBM_OBJECT_KEYS.Value = &KEYSTRING;

/*============== VERB =========================*/
/*  verb determination uses variable &IBMVERB */
Evaluate %Mode
When = "A"
   &IBMVERB = "Create";
   Break;
When = "U"
   &IBMVERB = "Update";
   Break;
When = "L"
   &IBMVERB = "Update";
   Break;
When = "C"
   &IBMVERB = "Update";
   Break;
When-Other
   &IBMVERB = "Retrieve";
End-Evaluate;

&IBMREC.IBM_OBJECT_VERB.Value = &IBMVERB;

/*  =====================  EVENT_ID GEN   ============================== */
/*  create event_id */

&NEWNUM = GetNextNumber(IBM_FETCH_ID.IBM_NEXT_EVENT_ID, 99999);
```

```
   /*  only use newnum if no error generating next number  */

   If &NEWNUM > 0 Then
      &IBMREC.IBM_EVENT_ID.Value = &NEWNUM;
   Else
      &IBMREC.IBM_EVENT_ID.Value = %Datetime;
   End-If;

   &IBMREC.IBM_EVENT_DTTM.Value = %Datetime;

   /* ==============  EVENT_STATUS  =================*/
   /* Validate and set event status &IBMSTATUS - list values if date is ok*/
   &IBMREC.IBM_EVENT_STATUS.Value = "0";

   /*================  INSERT EVENT INTO IBM_EVENT_TBL ============*/
   /*  insert row into table using record object*/


   &IBMREC.IBM_OBJECT_NAME.Value = &BO;

   &IBMREC.Insert();

End-Function;
```

Configuration properties for the WebSphere Adapter for PeopleSoft Enterprise

WebSphere Adapter for PeopleSoft Enterprise has several categories of configuration properties, which you set with the Adapter Connection wizard when it generates or creates objects and services.

You can change the resource adapter, managed connection factory, and activation specification properties in WebSphere Message Broker.

For more information, see the following topics.

- "PeopleSoft connection properties for the Adapter Connection wizard"
- "Resource adapter properties (PeopleSoft)" on page 1434
- "Managed connection factory properties (PeopleSoft)" on page 1436
- "Activation specification properties (PeopleSoft)" on page 1439
- "Interaction specification properties (PeopleSoft)" on page 1444

PeopleSoft connection properties for the Adapter Connection wizard:

Connection properties for the Adapter Connection wizard are used to establish a connection between the Adapter Connection wizard and the application from which the wizard obtains metadata. These properties specify such things as connection configuration, and logging options.

If you set any of these connection properties using bidirectional script, you must set values that identify the format of the bidirectional script entered for that property.

The connection properties and their purpose are described in the following table. A complete description of each property is provided in the sections that follow the table.

Table 248. Connection properties

Property name	Description
"Component interface jar file"	The PeopleSoft Enterprise component interface that the adapter uses to establish a connection to the PeopleSoft components that are targets of integration transactions
"Host name "	The name or address of the server that hosts PeopleSoft Enterprise
"Password " on page 1433	The password of the user account of the adapter on the PeopleSoft Enterprise server
"Port number" on page 1433	The port number at which PeopleSoft Enterprise is configured to listen for client requests.
"Prefix for business object names" on page 1433	A prefix to be added to generated business objects
"User name" on page 1433	The name of the user account that the adapter uses on the PeopleSoft Enterprise server

Component interface jar file

This property specifies the PeopleSoft Enterprise component interface that the adapter uses to establish a connection to the PeopleSoft components that are targets of integration transactions.

Table 249. Component interface jar file details

Required	Yes
Default	No default
Property type	String
Usage	The name of the JAR file that the adapter uses to connect to the PeopleSoft Enterprise components of interest
Example	`CWYES_PeopleSoft\connectorModule\WbiEvent.jar`
Globalized	No
Bidi supported	No

Host name

This property specifies the name or address of the server that hosts PeopleSoft Enterprise.

Table 250. Host name details

Required	Yes
Default	No default value
Property type	String
Usage	Identifies the server, either by name or IP address, that hosts PeopleSoft Enterprise
Example	`9.26.248.202`
Globalized	No
Bidi supported	No

Password

This property specifies the password of the user account of the adapter on the PeopleSoft Enterprise server.

Table 251. Password details

Required	Yes
Default	No default value
Property type	String
Usage	The restrictions (case, length, and character) are determined by the PeopleSoft Enterprise version.
Globalized	Yes
Bidi supported	Yes

Port number

This property specifies the port number at which PeopleSoft Enterprise is configured to listen for client requests.

Table 252. Port number details

Required	Yes
Default	The port number that is entered when you run the Adapter Connection wizard
Property type	Integer
Example	9000
Globalized	No
Bidi supported	No

Prefix for business object names

This property specifies a prefix to be added to generated business objects.

Table 253. Prefix details

Required	No
Default	No default
Property type	String
Usage	Use this property to distinguish between different business objects that are generated against the same PeopleSoft component interface.
Example	If you use `IB` as a prefix, all business objects that are generated by this service are named using this prefix.
Globalized	Yes
Bidi supported	No

User name

This property specifies the name of the user account that the adapter uses on the PeopleSoft Enterprise server.

Table 254. User name details

Required	Yes
Default	No default value
Property type	String
Usage	The restrictions (case, length, and character) are determined by the PeopleSoft Enterprise version.
Example	`DV1`
Globalized	Yes
Bidi supported	Yes

Resource adapter properties (PeopleSoft):

The resource adapter properties control the general operation of the adapter, such as specifying the namespace for business objects. You set the resource adapter properties using the Adapter Connection wizard when you configure the adapter.

The following table lists the resource adapter properties and their purpose. A complete description of each property is provided in the sections that follow the table.

Table 255. Resource adapter properties for the Adapter for PeopleSoft Enterprise

Property	**Description**
Adapter ID to use for logging and tracing	The adapter instance for CEI and PMI events with respect to logging and tracing.
LogFileMaxSize	Supported for compatibility with earlier versions
LogFilename	Supported for compatibility with earlier versions
LogNumberOfFiles	Supported for compatibility with earlier versions
TraceFileMaxSize	Supported for compatibility with earlier versions
TraceFileName	Supported for compatibility with earlier versions
TraceNumberOfFiles	Supported for compatibility with earlier versions

Adapter ID to use for logging and tracing (AdapterID)

This property identifies a specific deployment, or instance, of the adapter.

Table 256. Adapter ID to use for logging and tracing details

Required	Yes
Default	CWYMY_Adapter Without local transaction support: CWYAP_SAPAdapter With local transaction support: CWYAP_SAPAdapter_Tx
Property type	String
Usage	Use this property to identify the adapter instance for PMI events. If you are deploying multiple instances of an adapter, set this property to a unique value for each adapter instance. For inbound processing, this property is retrieved from the resource adapter properties. For outbound processing, it is retrieved from the managed connection factory properties.
Globalized	Yes
Bidi supported	No

Log file maximum size (LogFileMaxSize)

This property specifies the size of the log files in kilobytes.

Table 257. Log file maximum size details

Required	No
Default	`0`
Property type	Integer
Usage	When the log file reaches it maximum size, the adapter starts to use a new log file. If the file size is specified as `0`, or no maximum size is specified, the file does not have a maximum size.
Globalized	Yes
Bidi supported	No

Log file name (LogFilename)

This property specifies the full path name of the log file.

Table 258. Log file name details

Required	No
Default	No default value
Property type	String
Usage	This property is deprecated.
Globalized	Yes
Bidi supported	Yes

Log number of files (LogNumberOfFiles)

This property specifies the number of log files.

Table 259. Log number of files details

Required	No
Default	`1`
Property type	Integer
Usage	When a log file reaches its maximum size, the adapter starts to use another log file. If no value is specified, the adapter creates a single log file.
Globalized	Yes
Bidi supported	No

Trace file maximum size (TraceFileMaxSize)

This property specifies the size of the trace files in kilobytes.

Table 260. Trace file maximum size details

Required	No
Default	`0`
Property type	Integer
Usage	If no value is specified, the trace file has no maximum size.

Table 260. Trace file maximum size details (continued)

Globalized	Yes
Bidi supported	No

Trace file name (TraceFilename)

This property specifies the full path of the trace file.

Table 261. Trace file name details

Required	No
Default	No default value
Unit of measure	Kilobytes
Property type	String
Usage	This property is deprecated.
Globalized	Yes
Bidi supported	Yes

Trace number of files (TraceNumberOfFiles)

This property specifies the number of trace files to use. When a trace file reaches its maximum size, the adapter starts to use another trace file.

Table 262. Trace number of files details

Required	No
Default	1
Property type	Integer
Usage	If no value is specified, the adapter uses a single trace file.
Globalized	Yes
Bidi supported	No

Managed connection factory properties (PeopleSoft):

The adapter uses managed connection factory properties at run time to create an outbound connection instance with the PeopleSoft Enterprise server.

The following table lists and describes the managed connection factory properties for outbound communication. You set the managed connection factory properties using the Adapter Connection wizard.

A more detailed description of each property is provided in the sections that follow the table.

Table 263. Managed connection factory properties

Property	Description
Component interface for testing failed connection	The component interface that the adapter uses to validate a connection to the PeopleSoft Enterprise server
Host name	The name or address of the server that hosts PeopleSoft Enterprise

Table 263. Managed connection factory properties (continued)

Property	Description
"Language (Language)"	The language code that the adapter uses to log on to the PeopleSoft Enterprise server
"Password (Password)" on page 1438	The password of the user account of the adapter on the PeopleSoft Enterprise server
"Port number (Port)" on page 1438	The port number that the adapter uses to access the PeopleSoft Enterprise server
"User name (UserName)" on page 1438	The name of the user account that the adapter uses on the PeopleSoft Enterprise server

Component interface for testing failed connection (PingCompInterface)

This property specifies the name of the PeopleSoft Enterprise component interface that the adapter uses to validate a connection to the PeopleSoft Enterprise server.

Table 264. Component interface for testing failed connection details

Required	Yes
Default	The name of the first component interface in the list
Property type	String
Usage	Specify a component interface name that already exists within your PeopleSoft Enterprise applications.
Example	`WBI_CUSTOMER_CI`
Globalized	No
Bidi supported	No

Host name (HostName)

This property specifies the name or address of the server that hosts PeopleSoft Enterprise.

Table 265. Host name details

Required	Yes
Default	No default value
Property type	String
Usage	Identifies, either by name or IP address, the server that hosts PeopleSoft Enterprise
Example	`9.26.248.202`
Globalized	No
Bidi supported	No

Language (Language)

This property specifies the language code that the adapter uses to log on to the PeopleSoft Enterprise server.

Table 266. Language details

Required	Yes
Default	The default value for the Language property is based on the system locale.

Table 266. Language details (continued)

Property type	String
Usage	Each of the supported languages is preceded by a three-character language code. The language itself is presented in parentheses.
Example	If the system locale is English, the value for this property is `ENG (English)`.
Globalized	No
Bidi supported	No

Password (Password)

This property specifies the password of the user account of the adapter on the PeopleSoft Enterprise server.

Table 267. Password details

Required	Yes
Default	No default value
Property type	String
Usage	The restrictions (case, length, and character) are determined by the PeopleSoft Enterprise version.
Globalized	No
Bidi supported	No

Port number (Port)

This property specifies the port number that the adapter uses to access the PeopleSoft Enterprise server.

Table 268. Port number details

Required	Yes
Default	The port number that is entered when you use the Adapter Connection wizard to discover objects and services
Property type	Integer
Example	`9000`
Globalized	No
Bidi supported	No

User name (UserName)

This property specifies the name of the user account that the adapter uses on the PeopleSoft Enterprise server.

Table 269. User name details

Required	Yes
Default	No default value
Property type	String
Usage	The restrictions (case, length, and character) are determined by the PeopleSoft Enterprise version.

Table 269. User name details (continued)

Example	DV1
Globalized	No
Bidi supported	No

Activation specification properties (PeopleSoft):

Activation specification properties hold the configuration information for inbound event processing of an export component. You set activation specification properties using the Adapter Connection wizard.

The following table lists the activation specification properties for inbound communication. A more detailed description of each property is provided in the sections that follow the table.

Table 270. Activation specification properties

Property name	Purpose
AssuredOnceDelivery	Specifies whether the adapter provides assured once-only delivery of events
PingCompIntfc	The component interface that the adapter uses to validate a connection to the PeopleSoft Enterprise server
EventCIName	The component interface that the adapter uses for event notification
DeliveryType	Determines the order in which events are delivered by the adapter to the export component
EventKeyDelimiter	The name and value for an object key in the event table
EventTypeFilter	A delimited list of event types that indicates to the adapter which events it should deliver
DateFormat	The format that is used to create the event timestamp
MaximumConnections	The maximum number of connections that the adapter can use for inbound event delivery
MinimumConnections	The minimum number of connections that the adapter can use for inbound event delivery
PollPeriod	The length of time that the adapter waits between polling periods
PollQuantity	The number of events that the adapter delivers to the export component during each poll period
RetryInterval	The length of time that the adapter waits between attempts to establish a new connection after an error occurs during inbound operations
RetryLimit	The number of times that the adapter tries to reestablish an inbound connection after an error occurs.
StopPollingOnError	Specifies whether the adapter stops polling for events when it encounters an error during polling.

Ensure once-only event delivery (AssuredOnceDelivery)

This property specifies whether to provide ensured once-only event delivery for inbound events.

Table 271. Ensure once-only event delivery details

Required	Yes

Table 271. Ensure once-only event delivery details (continued)

Possible values	`True` `False`
Default	`True`
Property type	Boolean
Usage	When this property is set to `True`, the adapter provides assured once-only event delivery so that each event is delivered only once. A value of `False` does not provide assured once event delivery, but provides better performance. When this property is set to `True`, the adapter attempts to store transaction (XID) information in the event store. If it is set to `False`, the adapter does not attempt to store the information. This property is used only if the export component is transactional. If it is not, no transaction can be used, regardless of the value of this property.
Globalized	No
Bidi supported	No

Component interface for testing failed connection (PingCompInterface)

This property specifies the name of the PeopleSoft Enterprise component interface that the adapter uses to validate a connection to the PeopleSoft Enterprise server.

Table 272. Component interface for testing failed connection details

Row	**Explanation**
Required	Yes
Default	The name of the first component interface in the list
Property type	String
Usage	The name of the component interface that the adapter uses to test connectivity to the PeopleSoft Enterprise server. Specify a component interface name that already exists in your PeopleSoft Enterprise applications.
Globalized	No
Bidi supported	No

Component interface name for event notification (EventCIName)

This property specifies the name of the PeopleSoft Enterprise component interface that the adapter uses to for inbound processing.

Table 273. Component interface name for event notification details

Row	**Explanation**
Required	Yes
Default	`IBM_EVENT_CI`
Property type	String
Usage	The name of the component interface that the adapter uses for inbound processing. To use inbound processing, create a component interface specifically for event notification in PeopleSoft Enterprise.
Globalized	No
Bidi supported	No

Delivery type (DeliveryType)

This property specifies the order in which events are delivered by the adapter to the export component.

Table 274. Delivery type details

Required	No
Possible values	`ORDERED` `UNORDERED`
Default	`ORDERED`
Property type	String
Usage	The following values are supported: • `ORDERED`: The adapter delivers events to the export component one at a time. • `UNORDERED`: The adapter delivers all events to the export component at once.
Globalized	No
Bidi supported	No

Delimiter for keys in the event store (EventKeyDelimiter)

This property specifies the delimiter for the object key name-value pair in the event table.

Table 275. Delimiter for keys in the event store details

Row	**Explanation**
Required	No
Default	`=:`
Property type	String
Usage	Use this property to specify an object name and value to be used as an object key in the event store.
Example	`CustomerID=2001`
Globalized	No
Bidi supported	No

Event types to process (EventTypeFilter)

This property contains a delimited list of event types that indicates to the adapter which events it should deliver.

Table 276. Event types to process details

Required	No
Possible values	A comma-delimited (,) list of business object types
Default	`null`
Property type	String
Usage	Events are filtered by business object type. If the property is set, the adapter delivers only those events that are in the list. A value of `null` indicates that no filter will be applied and that all events will be delivered to the export component.

Table 276. Event types to process details (continued)

Example	To receive only events that relate to the Customer and Order business objects, specify this value: `Customer,Order`
Globalized	No
Bidi supported	No

Java date format for event timestamp (DateFormat)

This property specifies the format that is used for the event timestamp.

Table 277. Java date format for event timestamp details

Row	Explanation
Required	Yes
Default	`MM/dd/yy`
Property type	String
Usage	This property is used to format the date values from the PeopleSoft Enterprise server.
Globalized	No
Bidi supported	No

Maximum connections (MaximumConnections)

This property specifies the maximum number of connections that the adapter can use for inbound event delivery.

Table 278. Maximum connections details

Required	No
Default	1
Property type	Integer
Usage	Only positive values are valid. The adapter considers any positive entry less than 1 to be equal to 1. Typing a negative value or 0 for this property might result in runtime errors.
Globalized	No
Bidi supported	No

Minimum connections (MinimumConnections)

This property specifies the minimum number of connections that the adapter can use for inbound event delivery.

Table 279. Minimum connections details

Required	No
Default	1
Property type	Integer
Usage	Only positive values are valid. Any value less than 1 is treated as 1 by the adapter. Typing a negative value or 0 for this property might result in runtime errors.
Globalized	No

Table 279. Minimum connections details (continued)

Bidi supported	No

Interval between polling periods (PollPeriod)

This property specifies the length of time that the adapter waits between polling periods.

Table 280. Interval between polling periods details

Required	Yes
Possible values	Integers greater than or equal to `0`.
Default	`2000`
Unit of measure	Milliseconds
Property type	Integer
Usage	The poll period is established at a fixed rate, which means that if running the poll cycle is delayed for any reason (for example, if a prior poll cycle takes longer than expected to complete) the next poll cycle occurs immediately to make up for the lost time that was caused by the delay.
Globalized	No
Bidi supported	No

Maximum events in polling period (PollQuantity)

This property specifies the number of events that the adapter delivers to the export component during each poll period.

Table 281. Maximum events in polling period details

Required	Yes
Default	`10`
Property type	Integer
Usage	The value must be greater than 0. If this value is increased, more events are processed per polling period, and the adapter might perform less efficiently. If this value is decreased, fewer events are processed per polling period, and the adapter's performance might improve slightly.
Globalized	No
Bidi supported	No

Retry interval if connection fails (RetryInterval)

When the adapter encounters an error that is related to the inbound connection, this property specifies the length of time that the adapter waits before trying to establish a new connection.

Table 282. Retry interval details

Required	Yes
Default	`2000`
Unit of measure	Milliseconds
Property type	Integer

Table 282. Retry interval details (continued)

Usage	Only positive values are valid. When the adapter encounters an error that is related to the inbound connection, this property specifies the length of time that the adapter waits before trying to establish a new connection.
Globalized	Yes
Bidi supported	No

Number of times to retry the system connection (RetryLimit)

This property specifies the number of times that the adapter tries to reestablish an inbound connection.

Table 283. Number of times to retry the system connection details

Required	No
Possible values	Positive integers
Default	`0`
Property type	Integer
Usage	Only positive values are valid. When the adapter encounters an error that is related to the inbound connection, this property specifies the number of times that the adapter tries to restart the connection. A value of `0` indicates an infinite number of retries.
Globalized	Yes
Bidi supported	No

Stop the adapter when an error is encountered while polling (StopPollingOnError)

This property specifies whether the adapter stops polling for events when it encounters an error during polling.

Table 284. Stop the adapter when an error is encountered while polling details

Required	No
Possible values	`True` `False`
Default	`False`
Property type	Boolean
Usage	If this property is set to `True`, the adapter stops polling when it encounters an error. If this property is set to `False`, the adapter logs an exception when it encounters an error during polling and continues polling.
Globalized	No
Bidi supported	No

Interaction specification properties (PeopleSoft):

Interaction specification properties control the interaction for an operation. The Adapter Connection wizard sets the interaction specification properties when you configure the adapter.

Typically, you do not need to change these properties. However, you can change some properties for outbound operations. For example, you can increase the value of the interaction specification property that specifies the maximum number of records to be returned by a RetrieveAll operation, if your RetrieveAll operations do not return complete information.

The following table lists and describes the interaction specification property that you set.

Table 285. Interaction specification property for the Adapter for PeopleSoft Enterprise

Property name	Description
Maximum number of records for RetrieveAll operation	The maximum number of records to return during a RetrieveAll operation

Maximum number of records for RetrieveAll operation (MaxRecords)

This property specifies the maximum number of records to return during a RetrieveAll operation.

Table 286. Maximum number of records for RetrieveAll operation details

Required	Yes
Default	100
Usage	If the number of hits in PeopleSoft Enterprise exceeds the value of the Maximum number of records for the RetrieveAll operation property, the adapter returns an error. The adapter uses this property to help avoid out-of-memory issues.
Property type	Integer
Globalized	No
Bidi supported	No

Validation properties

You can control validation by setting properties on the Validate and Parser Options tabs for the nodes that are listed in the following table.

Validation options are available on the following nodes:

Node type	Nodes with validation options
Input node	FileInput, HTTPInput, JMSInput, MQInput, SCADAInput, SOAPInput,TimeoutNotification,
Output node	FileOutput, HTTPReply, JMSOutput, JMSReply, MQOutput, MQReply, SCADAOutput, SOAPReply
Other nodes	Compute, , DatabaseRetrieve, HTTPRequest, JavaCompute, Mapping, MQGet, ResetContentDescriptor, SOAPRequest, SOAPAsyncResponse, Validate, XSLTransform

For an overview of message validation in the broker, refer to “Validating messages” on page 204.

You can set the properties that are shown in the following table.

Tab	Properties that affect validation
Validation	Validate, Failure Action
Parser Options	Parse Timing

Validation tab properties

Validate

Sets whether validation is required. All nodes provide the following options:

None The default value. No validation is performed.

Content
Indicates that you want to perform content checks, such as Content validation and Composition.

Content and Value
Indicates that you want to perform content checks, such as Content validation and Composition, and value checks, such as whether the value conforms to data type, length, range, and enumeration.

Note: Even if **Content** is selected, the SOAP and XMLNSC domains always perform **Content and Value** validation.

Some nodes also provide the following option:

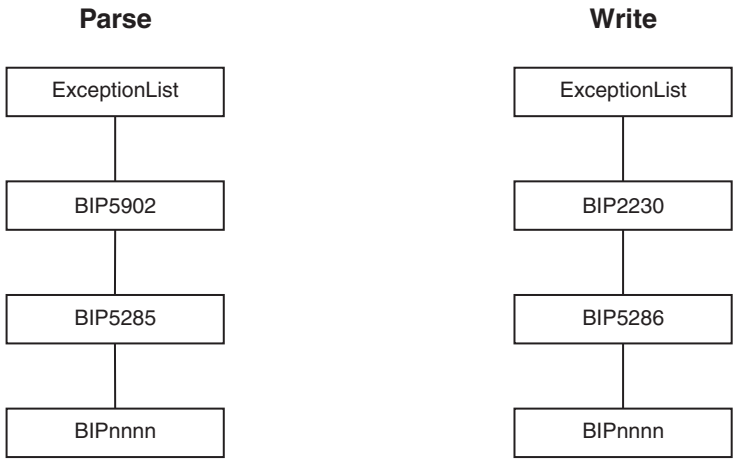
Inherit
Instructs the node to use all the validation options that are provided with the input message tree in preference to any supplied on the node. Inherit therefore resolves to None, Content, or Content And Value. If Inherit is selected, the other validation properties on the tab are not available.

Failure Action

The action that you want to be taken when a validation failure occurs. You can set it to the following values:

Exception
The default value. An exception is thrown on the first validation failure encountered. The resulting exception list is shown below. The failure is also logged in the user trace if you have asked for user tracing of the message flow, and validation stops. Use this setting if you want processing of the message to halt as soon as a failure is encountered.

MRM and IDOC

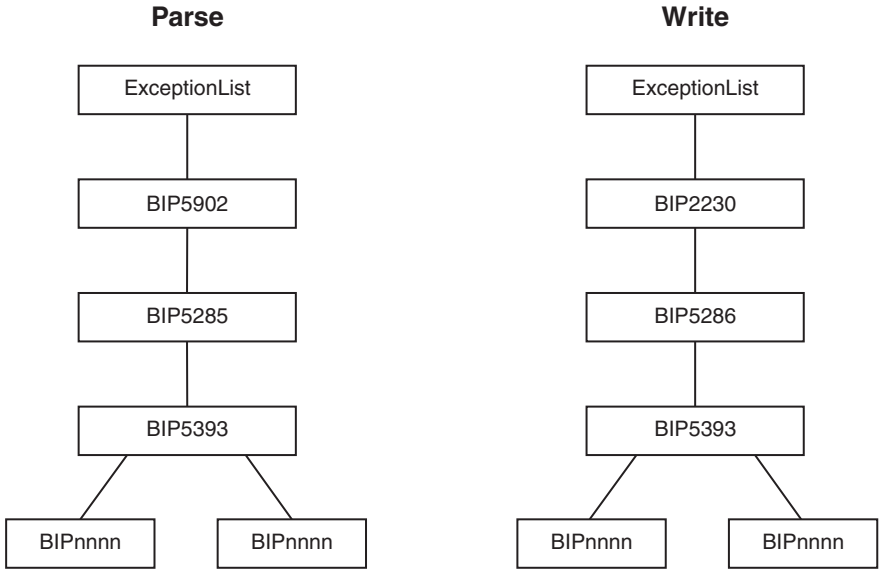


XMLNSC and SOAP

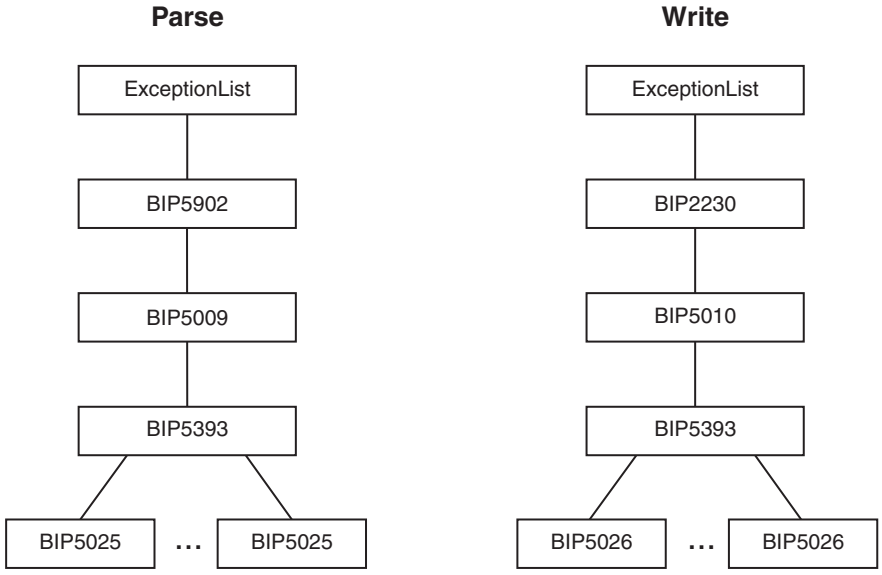
Exception List

Throws an exception if validation failures are encountered, but only when the current parsing or writing operation has completed. The resulting exception list is shown below. Each failure is also logged in the user trace if you have asked for user tracing of the message flow, and validation stops. Use this setting if you want processing of the message to halt if a validation failure occurs, but you want to see the full list of failures encountered. This property is affected by the Parse Timing property; when partial parsing is selected the current parsing operation parses only a portion of an input message, so only the validation failures in that portion of the message are reported.

MRM and IDOC



XMLNSC and SOAP



User Trace

Logs all validation failures to the user trace, even if you have not asked for user tracing of the message flow. Use this setting if you want processing of the message to continue regardless of validation failures.

Local Error Log

Logs all validation failures to the error log (for example, the Event Log on Windows). Use this setting if you want processing of the message to continue regardless of validation failures.

Parser Options tab properties

Parse Timing

The Parse Timing property determines whether on-demand parsing is to be used when parsing a message. It also gives you control over the timing of input message validation:

- If you select a Parse Timing value of On Demand, validation of a field in the message is delayed until it is parsed by on-demand parsing.
- If you select a Parse Timing value of Immediate, on-demand parsing is overridden, and everything in the message is parsed and validated except, if the message domain is MRM, those complex types with a Composition of Choice or Message that cannot be resolved at the time
- If you select a Parse Timing value of Complete, on-demand parsing is overridden, and everything is parsed and validated. If the message domain is MRM, complex types with a Composition of Choice or Message that cannot be resolved at the time cause a validation failure.

If you enable message validation, and you select On Demand or Immediate for Parse Timing, validation errors might not be detected until later in the processing of a message by a message flow, or might never be detected if a portion of the message is never parsed. To make sure that all fields in a message are validated, either select Complete or, if the message domain is MRM, select Immediate and make sure that you resolve all unresolved types with a Composition of Choice or Message at the start of your message flow.

The Parse Timing property does not affect the validation of output messages.

Parsing on demand

On-demand parsing, referred to as partial parsing, is used to parse an input message bit stream only as far as is necessary to satisfy the current reference. The parsers that are capable of performing partial parsing of input messages are the MRM, XML, XMLNS, and XMLNSC parsers.

An input message can be of any length. To improve performance of message flows, a message is parsed only when necessary to resolve the reference to a particular part of its content. If none of the message content is referenced within the message flow (for example, the entire message is stored in a database by the DataUpdate node, but no manipulation of the message content takes place), the message body is not parsed.

If a parser is capable of parsing an input bit stream on demand, instead of immediately parsing the entire bit stream, the Parse Timing property of a message flow node controls the on demand behavior of the parser.

You can set the Parse Timing property to On Demand (the default), Immediate, or Complete.

On Demand causes partial parsing to occur. When fields in the message are referenced, as much of the message is parsed as is necessary to completely resolve the reference. Therefore, fields might not be parsed until late in the message flow, or never. This restriction applies to both the message body and the message headers.

Immediate and Complete both override partial parsing and parse the entire message, including any message headers, except when the MRM parser encounters an element with a complex type with Composition set to Choice or Message that

cannot be resolved at the time; for example, the content needs to be resolved by the user in ESQL. If Composition is set to Choice, the data is added to the message tree as an unresolved item, and parsing continues with the next element. If Composition is set to Message, parsing terminates at that point. The only difference in behavior between Immediate and Complete occurs when MRM validation is enabled.

The Parse Timing property also gives you control over how MRM message validation interacts with partial parsing. Refer to "Validation properties" on page 1445 for a full description.

The Parse Timing property has no effect on the serialization of output messages.

User-defined nodes

You can define your own nodes to use in WebSphere Message Broker message flows.

User-defined nodes add to the function that is provided by the WebSphere Message Broker built-in nodes. You can also use nodes that are created and supplied by independent software vendors and other companies.

Follow the instructions in Adding help to the node to provide help information for user-defined nodes, and how to include that help in this section of the information center.

Supported code pages

Application messages must conform to supported code pages.

The message flows that you create, configure, and deploy to a broker can process and construct application messages in any code page that is listed in the following tables. You can also generate a new code page converter.

This behavior might be affected by the use of other products with WebSphere Message Broker. Check the documentation for other products, including any databases that you use, for further code page support information.

If you experience code page translation problems on HP-UX, check the WebSphere MQ queue manager attribute *CodedCharSetID* (CCSID). The default value for this attribute is 1051. Change this attribute value to 819 for queue managers that host WebSphere Message Broker components.

For detailed information about Chinese code page GB18030 support, see "Chinese code page GB18030" on page 1477.

By default, WebSphere Message Broker supports the code pages that are given in the following tables. To find a code page for a specific CCSID, search for an internal converter name in the form `ibm-ccsid`, where *ccsid* is the CCSID for which you are looking.

- Unicode converters
- European and American language converters
- Asian language converters
- Windows US and European converters
- MAC-related converters

- Hebrew, Cyrillic and ECMA language converters
- Indian language converters
- EBCDIC converters

Unicode converters

Internal converter name	Aliases
UTF-8	UTF-8 ibm-1208 ibm-1209 ibm-5304 ibm-5305 windows-65001 cp1208
UTF-16	UTF-16 ISO-10646-UCS-2 unicode csUnicode ucs-2
UTF-16BE	UTF-16BE x-utf-16be ibm-1200 ibm-1201 ibm-5297 ibm-13488 ibm-17584 windows-1201 cp1200 cp1201 UTF16_BigEndian
UTF-16LE	UTF-16LE x-utf-16le ibm-1202 ibm-13490 ibm-17586 UTF16_LittleEndian windows-1200
UTF-32	UTF-32 ISO-10646-UCS-4 csUCS4 ucs-4
UTF-32BE	UTF-32BE UTF32_BigEndian ibm-1232 ibm-1233

Internal converter name	Aliases
UTF-32LE	UTF-32LE UTF32_LittleEndian ibm-1234
UTF16_PlatformEndian	UTF16_PlatformEndian
UTF16_OppositeEndian	UTF16_OppositeEndian
UTF32_PlatformEndian	UTF32_PlatformEndian
UTF32_OppositeEndian	UTF32_OppositeEndian
UTF-7	UTF-7 windows-65000
IMAP-mailbox-name	IMAP-mailbox-name
SCSU	SCSU
BOCU-1	BOCU-1 csBOCU-1
CESU-8	CESU-8

European and American language converters

Internal converter name	Aliases
ISO-8859-1	ISO-8859-1 ibm-819 IBM819 cp819 latin1 8859_1 csISOLatin1 iso-ir-100 ISO_8859-1:1987 l1 819
US-ASCII	US-ASCII ASCII ANSI_X3.4-1968 ANSI_X3.4-1986 ISO_646.irv:1991 iso_646.irv:1983 ISO646-US us csASCII iso-ir-6 cp367 ascii7 646 windows-20127

Internal converter name	Aliases
gb18030	gb18030 ibm-1392 windows-54936
ibm-367_P100-1995	ibm-367_P100-1995 ibm-367 IBM367
ibm-912_P100-1995	ibm-912_P100-1995 ibm-912 iso-8859-2 ISO_8859-2:1987 latin2 csISOLatin2 iso-ir-101 l2 8859_2 cp912 912 windows-28592
ibm-913_P100-2000	ibm-913_P100-2000 ibm-913 iso-8859-3 ISO_8859-3:1988 latin3 csISOLatin3 iso-ir-109 l3 8859_3 cp913 913 windows-28593
ibm-914_P100-1995	ibm-914_P100-1995 ibm-914 iso-8859-4 latin4 csISOLatin4 iso-ir-110 ISO_8859-4:1988 l4 8859_4 cp914 914 windows-28594

Internal converter name	Aliases
ibm-915_P100-1995	ibm-915_P100-1995 ibm-915 iso-8859-5 cyrillic csISOLatinCyrillic iso-ir-144 ISO_8859-5:1988 8859_5 cp915 915 windows-28595
ibm-1089_P100-1995	ibm-1089_P100-1995 ibm-1089 iso-8859-6 arabic csISOLatinArabic iso-ir-127 ISO_8859-6:1987 ECMA-114 ASMO-708 8859_6 cp1089 1089 windows-28596 ISO-8859-6-I ISO-8859-6-E
ibm-813_P100-1995	ibm-813_P100-1995 ibm-813 iso-8859-7 greek greek8 ELOT_928 ECMA-118 csISOLatinGreek iso-ir-126 ISO_8859-7:1987 8859_7 cp813 813 windows-28597

Internal converter name	Aliases
ibm-916_P100-1995	ibm-916_P100-1995 ibm-916 iso-8859-8 hebrew csISOLatinHebrew iso-ir-138 ISO_8859-8:1988 ISO-8859-8-I ISO-8859-8-E 8859_8 cp916 916 windows-28598
ibm-920_P100-1995	ibm-920_P100-1995 ibm-920 iso-8859-9 latin5 csISOLatin5 iso-ir-148 ISO_8859-9:1989 l5 8859_9 cp920 920 windows-28599 ECMA-128
ibm-921_P100-1995	ibm-921_P100-1995 ibm-921 iso-8859-13 8859_13 cp921 921
ibm-923_P100-1998	ibm-923_P100-1998 ibm-923 iso-8859-15 Latin-9 l9 8859_15 latin0 csisolatin0 csisolatin9 iso8859_15_fdis cp923 923 windows-28605

Asian language converters

Internal converter name	Aliases
ibm-942_P12A-1999	ibm-942_P12A-1999 ibm-942 ibm-932 cp932 shift_jis78 sjis78 ibm-942_VSUB_VPUA ibm-932_VSUB_VPUA
ibm-943_P15A-2003	ibm-943_P15A-2003 ibm-943 Shift_JIS MS_Kanji csShiftJIS windows-31j csWindows31J x-sjis x-ms-cp932 cp932 windows-932 cp943c IBM-943C ms932 pck sjis ibm-943_VSUB_VPUA
ibm-943_P130-1999	ibm-943_P130-1999 ibm-943 Shift_JIS cp943 943 ibm-943_VASCII_VSUB_VPUA
ibm-33722_P12A-1999	ibm-33722_P12A-1999 ibm-33722 ibm-5050 EUC-JP Extended_UNIX_Code_Packed_Format_for_Japanese csEUCPkdFmtJapanese X-EUC-JP eucjis windows-51932 ibm-33722_VPUA IBM-eucJP

Internal converter name	Aliases
ibm-33722_P120-1999	ibm-33722_P120-1999 ibm-33722 ibm-5050 cp33722 33722 ibm-33722_VASCII_VPUA
ibm-954_P101-2000	ibm-954_P101-2000 ibm-954 EUC-JP
ibm-1373_P100-2002	ibm-1373_P100-2002 ibm-1373 windows-950
windows-950-2000	windows-950-2000 Big5 csBig5 windows-950 x-big5
ibm-950_P110-1999	ibm-950_P110-1999 ibm-950 cp950 950
macos-2566-10.2	macos-2566-10.2 Big5-HKSCS big5hk HKSCS-BIG5
ibm-1375_P100-2003	ibm-1375_P100-2003 ibm-1375 Big5-HKSCS
ibm-1386_P100-2002	ibm-1386_P100-2002 ibm-1386 cp1386 windows-936 ibm-1386_VSUB_VPUA
windows-936-2000	windows-936-2000 GBK CP936 MS936 windows-936

Internal converter name	Aliases
ibm-1383_P110-1999	ibm-1383_P110-1999 ibm-1383 GB2312 csGB2312 EUC-CN ibm-eucCN hp15CN cp1383 1383 ibm-1383_VPUA
ibm-5478_P100-1995	ibm-5478_P100-1995 ibm-5478 GB_2312-80 chinese iso-ir-58 csISO58GB231280 gb2312-1980 GB2312.1980-0
ibm-964_P110-1999	ibm-964_P110-1999 ibm-964 EUC-TW ibm-eucTW cns11643 cp964 964 ibm-964_VPUA
ibm-949_P110-1999	ibm-949_P110-1999 ibm-949 cp949 949 ibm-949_VASCII_VSUB_VPUA
ibm-949_P11A-1999	ibm-949_P11A-1999 ibm-949 cp949c ibm-949_VSUB_VPUA
ibm-970_P110-1995	ibm-970_P110-1995 ibm-970 EUC-KR KS_C_5601-1987 windows-51949 csEUCKR ibm-eucKR KSC_5601 5601 ibm-970_VPUA

Internal converter name	Aliases
ibm-971_P100-1995	ibm-971_P100-1995 ibm-971 ibm-971_VPUA
ibm-1363_P11B-1998	ibm-1363_P11B-1998 ibm-1363 KS_C_5601-1987 KS_C_5601-1989 KSC_5601 csKSC56011987 korean iso-ir-149 5601 cp1363 ksc windows-949 ibm-1363_VSUB_VPUA
ibm-1363_P110-1997	ibm-1363_P110-1997 ibm-1363 ibm-1363_VASCII_VSUB_VPUA
windows-949-2000	windows-949-2000 windows-949 KS_C_5601-1987 KS_C_5601-1989 KSC_5601 csKSC56011987 korean iso-ir-149 ms949
ibm-1162_P100-1999	ibm-1162_P100-1999 ibm-1162
ibm-874_P100-1995	ibm-874_P100-1995 ibm-874 ibm-9066 cp874 TIS-620 tis620.2533 eucTH cp9066
windows-874-2000	windows-874-2000 TIS-620 windows-874 MS874

Windows US and European converters

Internal converter name	Aliases
ibm-437_P100-1995	ibm-437_P100-1995 ibm-437 IBM437 cp437 437 csPC8CodePage437 windows-437
ibm-850_P100-1995	ibm-850_P100-1995 ibm-850 IBM850 cp850 850 csPC850Multilingual windows-850
ibm-851_P100-1995	ibm-851_P100-1995 ibm-851 IBM851 cp851 851 csPC851
ibm-852_P100-1995	ibm-852_P100-1995 ibm-852 IBM852 cp852 852 csPCp852 windows-852
ibm-855_P100-1995	ibm-855_P100-1995 ibm-855 IBM855 cp855 855 csIBM855 csPCp855
ibm-856_P100-1995	ibm-856_P100-1995 ibm-856 cp856 856
ibm-857_P100-1995	ibm-857_P100-1995 ibm-857 IBM857 cp857 857 csIBM857 windows-857

Internal converter name	Aliases
ibm-858_P100-1997	ibm-858_P100-1997 ibm-858 IBM00858 CCSID00858 CP00858 PC-Multilingual-850+euro cp858
ibm-860_P100-1995	ibm-860_P100-1995 ibm-860 IBM860 cp860 860 csIBM860
ibm-861_P100-1995	ibm-861_P100-1995 ibm-861 IBM861 cp861 861 cp-is csIBM861 windows-861
ibm-862_P100-1995	ibm-862_P100-1995 ibm-862 IBM862 cp862 862 csPC862LatinHebrew DOS-862 windows-862
ibm-863_P100-1995	ibm-863_P100-1995 ibm-863 IBM863 cp863 863 csIBM863
ibm-864_X110-1999	ibm-864_X110-1999 ibm-864 IBM864 cp864 csIBM864

Internal converter name	Aliases
ibm-865_P100-1995	ibm-865_P100-1995 ibm-865 IBM865 cp865 865 csIBM865
ibm-866_P100-1995	ibm-866_P100-1995 ibm-866 IBM866 cp866 866 csIBM866 windows-866
ibm-867_P100-1998	ibm-867_P100-1998 ibm-867 cp867
ibm-868_P100-1995	ibm-868_P100-1995 ibm-868 IBM868 cp868 868 csIBM868 cp-ar
ibm-869_P100-1995	ibm-869_P100-1995 ibm-869 IBM869 cp869 869 cp-gr csIBM869 windows-869
ibm-878_P100-1996	ibm-878_P100-1996 ibm-878 KOI8-R koi8 csKOI8R cp878
ibm-901_P100-1999	ibm-901_P100-1999 ibm-901_P100-1999 ibm-901
ibm-902_P100-1999	ibm-902_P100-1999 ibm-902

Internal converter name	Aliases
ibm-922_P100-1999	ibm-922_P100-1999 ibm-922 cp922 922
ibm-4909_P100-1999	ibm-4909_P100-1999 ibm-4909
ibm-5346_P100-1998	ibm-5346_P100-1998 ibm-5346 windows-1250 cp1250
ibm-5347_P100-1998	ibm-5347_P100-1998 ibm-5347 windows-1251 cp1251
ibm-5348_P100-1997	ibm-5348_P100-1997 ibm-5348 windows-1252 cp1252
ibm-5349_P100-1998	ibm-5349_P100-1998 ibm-5349 windows-1253 cp1253
ibm-5350_P100-1998	ibm-5350_P100-1998 ibm-5350 windows-1254 cp1254
ibm-9447_P100-2002	ibm-9447_P100-2002 ibm-9447 windows-1255 cp1255
windows-1256-2000	windows-1256-2000 windows-1256 cp1256
ibm-9449_P100-2002	ibm-9449_P100-2002 ibm-9449 windows-1257 cp1257
ibm-5354_P100-1998	ibm-5354_P100-1998 ibm-5354 windows-1258 cp1258

Internal converter name	Aliases
ibm-1250_P100-1995	ibm-1250_P100-1995 ibm-1250 windows-1250
ibm-1251_P100-1995	ibm-1251_P100-1995 ibm-1251 windows-1251
ibm-1252_P100-2000	ibm-1252_P100-2000 ibm-1252 windows-1252
ibm-1253_P100-1995	ibm-1253_P100-1995 ibm-1253 windows-1253
ibm-1254_P100-1995	ibm-1254_P100-1995 ibm-1254 windows-1254
ibm-1255_P100-1995	ibm-1255_P100-1995 ibm-1255
ibm-5351_P100-1998	ibm-5351_P100-1998 ibm-5351 windows-1255
ibm-1256_P110-1997	ibm-1256_P110-1997 ibm-1256
ibm-5352_P100-1998	ibm-5352_P100-1998 ibm-5352 windows-1256
ibm-1257_P100-1995	ibm-1257_P100-1995 ibm-1257
ibm-5353_P100-1998	ibm-5353_P100-1998 ibm-5353 windows-1257
ibm-1258_P100-1997	ibm-1258_P100-1997 ibm-1258 windows-1258

MAC-related converters

Internal converter name	Aliases
macos-0_2-10.2	macos-0_2-10.2 macintosh mac csMacintosh windows-10000

Internal converter name	Aliases
macos-6-10.2	macos-6-10.2 x-mac-greek windows-10006 macgr
macos-7_3-10.2	macos-7_3-10.2 x-mac-cyrillic windows-10007 maccy
macos-29-10.2	macos-29-10.2 x-mac-centraleurroman windows-10029 x-mac-ce macce
macos-35-10.2	macos-35-10.2 x-mac-turkish windows-10081 mactr
ibm-1051_P100-1995	ibm-1051_P100-1995 ibm-1051 hp-roman8 roman8 r8 csHPRoman8
ibm-1276_P100-1995	ibm-1276_P100-1995 ibm-1276 Adobe-Standard-Encoding csAdobeStandardEncoding
ibm-1277_P100-1995	ibm-1277_P100-1995 ibm-1277 Adobe-Latin1-Encoding

Hebrew, Cyrillic, and ECMA language converters

Internal converter name	Aliases
ibm-1006_P100-1995	ibm-1006_P100-1995 ibm-1006 cp1006 1006
ibm-1098_P100-1995	ibm-1098_P100-1995 ibm-1098 cp1098 1098

Internal converter name	Aliases
ibm-1124_P100-1996	ibm-1124_P100-1996 ibm-1124 cp1124 1124
ibm-1125_P100-1997	ibm-1125_P100-1997 ibm-1125 cp1125
ibm-1129_P100-1997	ibm-1129_P100-1997 ibm-1129
ibm-1131_P100-1997	ibm-1131_P100-1997 ibm-1131 cp1131
ibm-1133_P100-1997	ibm-1133_P100-1997 ibm-1133
ibm-1381_P110-1999	ibm-1381_P110-1999 ibm-1381 cp1381 1381
ISO_2022,locale=ja,version=0	ISO_2022,locale=ja,version=0 ISO-2022-JP csISO2022JP
ISO_2022,locale=ja,version=1	ISO_2022,locale=ja,version=1 ISO-2022-JP-1 JIS JIS_Encoding
ISO_2022,locale=ja,version=2	ISO_2022,locale=ja,version=2 ISO-2022-JP-2 csISO2022JP2
ISO_2022,locale=ja,version=3	ISO_2022,locale=ja,version=3 JIS7 csJISEncoding
ISO_2022,locale=ja,version=4	ISO_2022,locale=ja,version=4 JIS8
ISO_2022,locale=ko,version=0	ISO_2022,locale=ko,version=0 ISO-2022-KR csISO2022KR
ISO_2022,locale=ko,version=1	ISO_2022,locale=ko,version=1 ibm-25546
ISO_2022,locale=zh,version=0	ISO_2022,locale=zh,version=0 ISO-2022-CN
ISO_2022,locale=zh,version=1	ISO_2022,locale=zh,version=1 ISO-2022-CN-EXT

Internal converter name	Aliases
HZ	HZ HZ-GB-2312
ibm-897_P100-1995	ibm-897_P100-1995 ibm-897 JIS_X0201 X0201 csHalfWidthKatakana

Indian language converters

Internal converter name	Aliases
ISCII,version=0 ISCII,version=0	x-iscii-de windows-57002 iscii-dev
ISCII,version=1 ISCII,version=1	x-iscii-be windows-57003 iscii-bng windows-57006 x-iscii-as
ISCII,version=2 ISCII,version=2	x-iscii-pa windows-57011 iscii-gur
ISCII,version=3 ISCII,version=3	x-iscii-gu windows-57010 iscii-guj
ISCII,version=4 ISCII,version=4	x-iscii-or windows-57007 iscii-ori
ISCII,version=5 ISCII,version=5	x-iscii-ta windows-57004 iscii-tml
ISCII,version=6 ISCII,version=6	x-iscii-te windows-57005 iscii-tlg
ISCII,version=7 ISCII,version=7	x-iscii-ka windows-57008 iscii-knd
ISCII,version=8 ISCII,version=8	x-iscii-ma windows-57009 iscii-mlm

EBCDIC converters

Internal converter name	Aliases
LMBCS-1	LMBCS-1 lmbcs
LMBCS-2	LMBCS-2
LMBCS-3	LMBCS-3
LMBCS-4	LMBCS-4
LMBCS-5	LMBCS-5
LMBCS-6	LMBCS-6
LMBCS-8	LMBCS-8
LMBCS-11	LMBCS-11
LMBCS-16	LMBCS-16
LMBCS-17	LMBCS-17
LMBCS-18	LMBCS-18
LMBCS-19	LMBCS-19
ibm-37_P100-1995	ibm-37_P100-1995 ibm-37 IBM037 ibm-037 ebcdic-cp-us ebcdic-cp-ca ebcdic-cp-wt ebcdic-cp-nl csIBM037 cp037 037 cpibm37 cp37
ibm-273_P100-1995	ibm-273_P100-1995 ibm-273 IBM273 CP273 csIBM273 ebcdic-de cpibm273 273

Internal converter name	Aliases
ibm-277_P100-1995	ibm-277_P100-1995 ibm-277 IBM277 cp277 EBCDIC-CP-DK EBCDIC-CP-NO csIBM277 ebcdic-dk cpibm277 277
ibm-278_P100-1995	ibm-278_P100-1995 ibm-278 IBM278 cp278 ebcdic-cp-fi ebcdic-cp-se csIBM278 ebcdic-sv cpibm278 278
ibm-280_P100-1995	ibm-280_P100-1995 ibm-280 IBM280 CP280 ebcdic-cp-it csIBM280 cpibm280 280
ibm-284_P100-1995	ibm-284_P100-1995 ibm-284 IBM284 CP284 ebcdic-cp-es csIBM284 cpibm284 284
ibm-285_P100-1995	ibm-285_P100-1995 ibm-285 IBM285 CP285 ebcdic-cp-gb csIBM285 ebcdic-gb cpibm285 285

Internal converter name	Aliases
ibm-290_P100-1995	ibm-290_P100-1995 ibm-290 IBM290 cp290 EBCDIC-JP-kana csIBM290
ibm-297_P100-1995	ibm-297_P100-1995 ibm-297 IBM297 cp297 ebcdic-cp-fr csIBM297 cpibm297 297
ibm-420_X120-1999	ibm-420_X120-1999 IBM420 cp420 ebcdic-cp-ar1 csIBM420 420
ibm-424_P100-1995	ibm-424_P100-1995 ibm-424 IBM424 cp424 ebcdic-cp-he csIBM424 424
ibm-500_P100-1995	ibm-500_P100-1995 ibm-500 IBM500 CP500 ebcdic-cp-be csIBM500 ebcdic-cp-ch cpibm500 500
ibm-803_P100-1999	ibm-803_P100-1999 ibm-803 cp803

Internal converter name	Aliases
ibm-838_P100-1995	ibm-838_P100-1995 ibm-838 IBM-Thai csIBMThai cp838 838 ibm-9030
ibm-870_P100-1995	ibm-870_P100-1995 ibm-870 IBM870 CP870 ebcdic-cp-roece ebcdic-cp-yu csIBM870
ibm-871_P100-1995	ibm-871_P100-1995 ibm-871 IBM871 ebcdic-cp-is csIBM871 CP871 ebcdic-is cpibm871 871
ibm-875_P100-1995	ibm-875_P100-1995 ibm-875 IBM875 cp875 875
ibm-918_P100-1995	ibm-918_P100-1995 ibm-918 IBM918 CP918 ebcdic-cp-ar2 csIBM918
ibm-930_P120-1999	ibm-930_P120-1999 ibm-930 ibm-5026 cp930 cpibm930 930

Internal converter name	Aliases
ibm-933_P110-1995	ibm-933_P110-1995 ibm-933 cp933 cpibm933 933
ibm-935_P110-1999	ibm-935_P110-1999 ibm-935 cp935 cpibm935 935
ibm-937_P110-1999	ibm-937_P110-1999 ibm-937 cp937 cpibm937 937
ibm-939_P120-1999	ibm-939_P120-1999 ibm-939 ibm-931 ibm-5035 cp939 939
ibm-1025_P100-1995	ibm-1025_P100-1995 ibm-1025 cp1025 1025
ibm-1026_P100-1995	ibm-1026_P100-1995 ibm-1026 IBM1026 CP1026 csIBM1026 1026
ibm-1047_P100-1995	ibm-1047_P100-1995 ibm-1047 IBM1047 cpibm1047
ibm-1097_P100-1995	ibm-1097_P100-1995 ibm-1097 cp1097 1097
ibm-1112_P100-1995	ibm-1112_P100-1995 ibm-1112 cp1112 1112

Internal converter name	Aliases
ibm-1122_P100-1999	ibm-1122_P100-1999 ibm-1122 cp1122 1122
ibm-1123_P100-1995	ibm-1123_P100-1995 ibm-1123 cp1123 1123 cpibm1123
ibm-1130_P100-1997	ibm-1130_P100-1997 ibm-1130
ibm-1132_P100-1998	ibm-1132_P100-1998 ibm-1132
ibm-1140_P100-1997	ibm-1140_P100-1997 ibm-1140 IBM01140 CCSID01140 CP01140 cp1140 cpibm1140 ebcdic-us-37+euro
ibm-1141_P100-1997	ibm-1141_P100-1997 ibm-1141 IBM01141 CCSID01141 CP01141 cp1141 cpibm1141 ebcdic-de-273+euro
ibm-1142_P100-1997	ibm-1142_P100-1997 ibm-1142 IBM01142 CCSID01142 CP01142 cp1142 cpibm1142 ebcdic-dk-277+euro ebcdic-no-277+euro

Internal converter name	Aliases
ibm-1143_P100-1997	ibm-1143_P100-1997 ibm-1143 IBM01143 CCSID01143 CP01143 cp1143 cpibm1143 ebcdic-fi-278+euro ebcdic-se-278+euro
ibm-1144_P100-1997	ibm-1144_P100-1997 ibm-1144 IBM01144 CCSID01144 CP01144 cp1144 cpibm1144 ebcdic-it-280+euro
ibm-1145_P100-1997	ibm-1145_P100-1997 ibm-1145 IBM01145 CCSID01145 CP01145 cp1145 cpibm1145 ebcdic-es-284+euro
ibm-1146_P100-1997	ibm-1146_P100-1997 ibm-1146 IBM01146 CCSID01146 CP01146 cp1146 cpibm1146 ebcdic-gb-285+euro
ibm-1147_P100-1997	ibm-1147_P100-1997 ibm-1147 IBM01147 CCSID01147 CP01147 cp1147 cpibm1147 ebcdic-fr-297+euro

Internal converter name	Aliases
ibm-1148_P100-1997	ibm-1148_P100-1997 ibm-1148 IBM01148 CCSID01148 CP01148 cp1148 cpibm1148 ebcdic-international-500+euro
ibm-1149_P100-1997	ibm-1149_P100-1997 ibm-1149 IBM01149 CCSID01149 CP01149 cp1149 cpibm1149 ebcdic-is-871+euro
ibm-1153_P100-1999	ibm-1153_P100-1999 ibm-1153 cpibm1153
ibm-1154_P100-1999	ibm-1154_P100-1999 ibm-1154 cpibm1154
ibm-1155_P100-1999	ibm-1155_P100-1999 ibm-1155 cpibm1155
ibm-1156_P100-1999	ibm-1156_P100-1999 ibm-1156 cpibm1156
ibm-1157_P100-1999	ibm-1157_P100-1999 ibm-1157 cpibm1157
ibm-1158_P100-1999	ibm-1158_P100-1999 ibm-1158 cpibm1158
ibm-1160_P100-1999	ibm-1160_P100-1999 ibm-1160 cpibm1160
ibm-1164_P100-1999	ibm-1164_P100-1999 ibm-1164 cpibm1164
ibm-1364_P110-1997	ibm-1364_P110-1997 ibm-1364 cp1364

Internal converter name	Aliases
ibm-1371_P100-1999	ibm-1371_P100-1999 ibm-1371 cpibm1371
ibm-1388_P103-2001	ibm-1388_P103-2001 ibm-1388 ibm-9580
ibm-1390_P110-2003	ibm-1390_P110-2003 ibm-1390 cpibm1390
ibm-1399_P110-2003	ibm-1399_P110-2003 ibm-1399
ibm-16684_P110-2003	ibm-16684_P110-2003 ibm-16684
ibm-4899_P100-1998	ibm-4899_P100-1998 ibm-4899 cpibm4899
ibm-4971_P100-1999	ibm-4971_P100-1999 ibm-4971 cpibm4971
ibm-12712_P100-1998	ibm-12712_P100-1998 ibm-12712 cpibm12712 ebcdic-he
ibm-16804_X110-1999	ibm-16804_X110-1999 ibm-16804 cpibm16804 ebcdic-ar
ibm-1137_P100-1999	ibm-1137_P100-1999 ibm-1137
ibm-5123_P100-1999	ibm-5123_P100-1999 ibm-5123
ibm-8482_P100-1999	ibm-8482_P100-1999 ibm-8482
ibm-37_P100-1995,swaplfnl	ibm-37_P100-1995,swaplfnl ibm-37-s390 ibm037-s390
ibm-1047_P100-1995,swaplfnl	ibm-1047_P100-1995,swaplfnl ibm-1047-s390
ibm-1140_P100-1997,swaplfnl	ibm-1140_P100-1997,swaplfnl ibm-1140-s390

Internal converter name	Aliases
ibm-1142_P100-1997,swaplfnl	ibm-1142_P100-1997,swaplfnl ibm-1142-s390
ibm-1143_P100-1997,swaplfnl	ibm-1143_P100-1997,swaplfnl ibm-1143-s390
ibm-1144_P100-1997,swaplfnl	ibm-1144_P100-1997,swaplfnl ibm-1144-s390
ibm-1145_P100-1997,swaplfnl	ibm-1145_P100-1997,swaplfnl ibm-1145-s390
ibm-1146_P100-1997,swaplfnl	ibm-1146_P100-1997,swaplfnl ibm-1146-s390
ibm-1147_P100-1997,swaplfnl	ibm-1147_P100-1997,swaplfnl ibm-1147-s390
ibm-1148_P100-1997,swaplfnl	ibm-1148_P100-1997,swaplfnl ibm-1148-s390
ibm-1149_P100-1997,swaplfnl	ibm-1149_P100-1997,swaplfnl ibm-1149-s390
ibm-1153_P100-1999,swaplfnl	ibm-1153_P100-1999,swaplfnl ibm-1153-s390
ibm-12712_P100-1998,swaplfnl	ibm-12712_P100-1998,swaplfnl ibm-12712-s390
ibm-16804_X110-1999,swaplfnl	ibm-16804_X110-1999,swaplfnl ibm-16804-s390
ebcdic-xml-us	ebcdic-xml-us

Chinese code page GB18030

If you are working with messages in Chinese code page GB18030, your messages might be subject to some restrictions.

The broker can input, manipulate, and output application messages that are encoded in code page IBM-5488 (GB18030 support) with the following restrictions:

- If you configure a message flow to store `GB18030` data in character form in a user database, ensure that the database manager that you are using supports `GB18030`.
- To enable support for `GB18030` in the workbench and Configuration Manager:
  - If you run a workbench or Configuration Manager that requires `GB18030` support on a computer that is running Windows 2003, apply the `GB18030` patch supplied by Microsoft. This support is included in Windows XP.
  - Change the text font preference in the workbench to use `GB18030`:
    - Select **Window** → **Preferences**.
    - Expand the **Workbench** item in the left pane of the Preferences dialog (click the plus sign), and select **Fonts**.
    - In the Fonts window, select **Text Font**. Click **Change**, and select the correct values in the Fonts selection dialog.

- Click **OK** to confirm the selection and close the dialog.
- Click **Apply** to apply the change, then click **OK** to close the Preference dialog.

WebSphere MQ connections

The number of WebSphere MQ connections a broker requires to its queue manager depends on the actions of the message flows that access the WebSphere MQ resource.

For each broker flow that accesses a queue, one connection is required for every message flow thread. If a different node on the same thread uses the same queue manager, the same connection is used.

The number of queue handles required also depends on the behavior of the flow. For each flow that accesses queues, one queue handle is required for each unique queue name for every message flow thread. Nodes that access the same queue name in the same flow use the same queue handle.

When you start a broker, and while it is running, it opens WebSphere MQ queue handles. The broker caches these queue handles. For example, when a message flow node initiates access to the first WebSphere MQ resource it uses, it opens a connection for the queue manager and opens the queue. This connection is opened the first time that a message is processed by that message flow node. For MQInput nodes, the connection is opened when the flow is started. This queue handle remains open until:

The message flow becomes idle, and has not been used for 1 minute

The execution group is stopped

The broker is stopped

The queue handle for the input node is not released when the flow is idle. The queue handle is released only when you stop the message flow.

A thread performing WebSphere MQ work becomes idle when it has not received any messages on its input queue for 1 minute. The allowed idle time starts from when the input queue being read becomes empty. If a message flow gets a message from the input queue, the timer is reset.

When a message flow is idle, the execution group periodically releases WebSphere MQ queue handles. Therefore, connections held by the broker reflect its current use of these resources.

Listing database connections that the broker holds

The broker does not provide an interface that you can use to list the connections that it has to a database. You must use the facilities of the database suppliers to list connections.

For further information about how to list database connections, refer to the documentation that is provided by your database vendor.

Quiescing a database

The broker and database exhibit specific behaviors when you quiesce the database.

If you access databases from one or more message flows, your database administrator might occasionally want to issue the quiesce instruction on a database. This action is a function of the database, not of the broker.

The following three assumptions are made for the database that you are quiescing:

- The database can be quiesced (not all databases support this function).
- New connections to the database are blocked by the database when it is quiescing.
- Message flows that access the database eventually become idle.

The following list shows the behavior that is expected while a database is quiescing:

- Run the command that quiesces the database. When this command starts, the connections that are in use remain in use, but no new connections to the database are allowed.
- Messages that are being processed by message flows, which use existing connections to the database, continue to use their connections until the connections become idle. Therefore if messages continue to be received by the message flow, it might be a long time before the quiesce occurs. To ensure that messages are no longer processed, stop the message flow. Stopping the message flow stops messages being processed, and releases the database connections that the flow was using. This action ensures that the database connections that the flow holds become idle.
- Database connections for the message flow become idle. This situation causes the broker to release the connections to the user databases that the message flow is using. When all connections to the database from the broker, and from any other applications that are using the database, are released, the database can complete its quiesce function.

For more information, see User database connections.

Support for Unicode and DBCS data in databases

You can manipulate Unicode Standard version 3.0 data, in suitably configured databases, using ESQL, in nodes that access databases by ODBC. The broker does not support DBCS-only columns in tables that are defined in databases.

The broker does not, therefore, support certain data types, including the following types:

- GRAPHIC, VARGRAPHIC, LONGVARGRAPHIC, DBCLOB (on DB2)
- NCHAR, NVARCHAR, NVARCHAR2, NCLOB (on Oracle)
- NCHAR, NVARCHAR, NTEXT, UNICHAR, UNIVARCHAR (on Sybase)
- NCHAR, NVARCHAR (on Informix®)

Support for Unicode is available only for the generally-supported versions of the following database managers:

- IBM DB2 v9 for Windows, Linux, UNIX, and z/OS operating systems.
- Oracle
- Microsoft SQL server
- Sybase Adaptive Server Enterprise (ASE)

Support for the manipulation of Unicode data is not available for nodes that access databases that use JDBC; for example, DatabaseRetrieve and DatabaseRoute.

The following instructions apply to both 32-bit and 64-bit applications.

If you are using DB2:

- On Windows, Linux, and UNIX operating systems, your database must be created with code set `utf-8`.
- On z/OS, set the variable MQSI_DB2_CONVERSION in the broker environment to the value `UNICODE`. In the ODBC definition add the statement `CURRENTAPPENSCH=UNICODE` to the `[COMMON]` stanza.
- On all platforms, DB2 returns the lengths of strings in bytes, rather than characters; this response has implications for the behavior of string length-related ESQL functions.

  Some functions might fail, or function differently, when processed by the database. See "Unicode string functions in DB2" for further information.

If you are using Oracle:

- Your database must be created with NLS_CHARACTERSET of AL32UTF8.
- Your ODBC data source definition must include the setting `ColumnSizeAsCharacter=1`.

  On UNIX and Linux platforms, this setting must be included in the appropriate stanza in the ODBC ini files.

  On Windows platforms, this string value must be added to the ODBC data source key in the registry.

  See Enabling ODBC connections to the databases for further information.
- For 32-bit connections, you must set the variable NLS_LANG in the broker environment to the value `<yourlanguage>_<yourterritory>.AL32UTF8`.

if you are using Microsoft SQL server:

- You must use NCHAR, NVARCHAR, and NTEXT data types for your column definitions.
- For brokers on UNIX and Linux platforms, your ODBC data source definition must include the setting `ColumnSizeAsCharacter=1`; this setting must be included in the appropriate stanza in the ODBC `.ini` files.

If you are using Sybase ASE:

- The default character set of your ASE server must be `UTF-8`.
- Your ODBC data source definition must include the settings `ColumnSizeAsCharacter=1` and `CharSet=UTF8`.

  On UNIX and Linux platforms, this setting must be included in the appropriate stanza in the ODBC `.ini` files.

  On Windows platforms, this string value must be added to the ODBC data source key in the registry.

  See Enabling ODBC connections to the databases for further information.

Unicode string functions in DB2

When you use string functions in ESQL expressions, certain parameters refer to character positions or counts.

For example, with the SUBSTRING function, coding:

```
SUBSTRING('Hello World!' FROM 7 FOR 4)
```

results in the string `Worl` because `7` refers to the seventh character position, and `4` refers to four characters.

If the string `Hello World!` is represented in an ASCII code page, the seventh *byte* of that representation is the seventh character. However, in other code pages or encoding schemes (for example, some Unicode representations) the seventh character could start at byte 13, and 4 characters can occupy 8 bytes.

WebSphere Message Broker correctly handles this situation; that is, the numeric parameters and results of these string functions always refer to *characters* and not the bytes used to represent them.

In some situations, WebSphere Message Broker delegates expressions to a database engine for processing. For example, if there is a WHERE clause, in a SELECT function, applied to a database data source, the database is passed the WHERE clause if it can interpret all the functions in the expression.

If there are functions that are not supported by the database, WebSphere Message Broker passes only those parts of the expression that can be interpreted, retrieves an unfiltered record set, and performs the remaining filtering itself.

DB2 string functions use *byte* indexing and not *character* indexing. Therefore, for Unicode data, the meaning of certain functions differs from the WebSphere Message Broker functions, even though they can be 'interpreted'.

Characters in Unicode UTF8 representation, for example, can occupy from 1-4 bytes, so that the seventh character can start anywhere from byte 7 to byte 25.

The following string functions are affected:

- INSERT function
- LEFT function
- LENGTH function
- OVERLAY function
- POSITION function
- RIGHT function
- SPACE function
- SUBSTRING function

These functions either take numeric parameters, or return numeric results that refer to indexes, or counts, of characters in a string.

When expressions involving these functions are passed to the DB2 database, and Unicode string data is manipulated in the database, the results can be unexpected, or an error might occur.

This error might also occur if, for example, an expression of this type is passed directly to the database by using the PASSTHRU function. In this situation, you could modify each expression yourself, as necessary, for the target database.

It is not possible to systematically modify expressions to avoid this problem and WebSphere Message Broker does not attempt to do so.

If the Unicode strings do not use any characters that, in UTF8 representation, occupy more than 1 byte each, the functions perform correctly.

Data integrity in message flows

Code pages in which data is manipulated must be compatible between brokers and databases that are accessed by those brokers.

Subscription data that is retrieved from client applications (for example, topics from publishers and subscribers, and content filters from subscribers), and the character data entered through the workbench (for example, message flow names), are stored in the configuration repository. This data is translated from its originating code page to the code page of the process in which the broker or Configuration Manager is running, then by the database manager to the code page in which the database or databases were created.

To preserve data consistency and integrity, ensure that all this subscription data and workbench character data is originated in a compatible code page to the two code pages to which it is translated. If you do not do so, you might get unpredictable results and lose data.

Data stored in the broker database is not affected in this way.

The restrictions described do not apply to user data in messages. Ensure that any data in messages generated by your applications is compatible with the code page of all databases that you access from your message flows.

SQL statements that are generated as a result of explicit reference to databases in message processing nodes can contain character data that has a variety of sources. For example, the data might have been entered through the workbench, derived from message content, or read from another database. All this data is translated from its originating code page to the code page in which the broker was created, then by the database manager to the code page in which the database was created. Ensure that these three code pages are compatible to avoid data conversion problems.

Exception list structure

The following figure shows one way in which to construct an exception list.

```
ExceptionList {
    RecoverableException = {                1
        File     = 'f:/build/argo/src/DataFlowEngine/ImbDataFlowNode.cpp'
        Line     = 538
        Function = 'ImbDataFlowNode::createExceptionList'
        Type     = 'ComIbmComputeNode'
        Name     = '0e416632-de00-0000-0080-bdb4d59524d5'
        Label    = 'mf1.Compute1'
        Text     = 'Node throwing exception'
        Catalog  = 'WebSphere Message Broker2'
        Severity = 3
        Number   = 2230
        RecoverableException = {              2
            File     = 'f:/build/argo/src/DataFlowEngine/ImbRdlBinaryExpression.cpp'
            Line     = 231
            Function = 'ImbRdlBinaryExpression::scalarEvaluate'
            Type     = 'ComIbmComputeNode'
            Name     = '0e416632-de00-0000-0080-bdb4d59524d5'
            Label    = 'mf1.Compute1'
            Text     = 'error evaluating expression'
            Catalog  = 'WebSphere Message Broker2'
            Severity = 2
            Number   = 2439
            Insert   = {
               Type = 2
               Text = '2'
            }
            Insert   = {
               Type = 2
               Text = '30'
            }
            RecoverableException = {             3
                File     = 'f:/build/argo/src/DataFlowEngine/ImbRdlValueOperations.cpp'
                Line     = 257
                Function = 'intDivideInt'
                Type     = 'ComIbmComputeNode'
                Name     = '0e416632-de00-0000-0080-bdb4d59524d5'
                Label    = 'mf1.Compute1'
                Text     = 'Divide by zero calculating '%1 / %2''
                Catalog  = 'WebSphere Message Broker2'
                Severity = 2
                Number   = 2450
                Insert   = }
                   Type = 5
                   Text = '100 / 0'
                }
            }
        }
    }
}
```

Notes:

1. The first exception description 1 is a child of the root. This identifies error number 2230, indicating that an exception has been thrown. The node that has thrown the exception is also identified (*mf1.Compute1*).
2. Exception description 2 is a child of the first exception description 1. This identifies error number 2439.
3. Exception description 3 is a child of the second exception description 2. This identifies error number 2450, which indicates that the node has attempted to divide by zero.

The following topics provide examples of exception lists that have been written to the trace output destination (by the Trace node):

Database exception trace output

The following figure shows an extract of the output that might be generated by a Trace node that has its property *Pattern* set to a value that represents a structure that includes the ExceptionList tree.

The exception shown occurred when a database exception was detected

```
ExceptionList = (
  (0x1000000)RecoverableException = (
    (0x3000000)File                = 'F:\build\S000_D\src\DataFlowEngine\ImbComputeNode.cpp'
    (0x3000000)Line                = 402
    (0x3000000)Function            = 'ImbComputeNode::evaluate'
    (0x3000000)Type                = 'ComIbmComputeNode'
    (0x3000000)Name                = 'acd8f35d-e700-0000-0080-b78796c5e70d'
    (0x3000000)Label               = 'esql_13485_check_defect.Compute1'
    (0x3000000)Text                = 'Caught exception and rethrowing'
    (0x3000000)Catalog             = 'WMQIv210'
    (0x3000000)Severity            = 3
    (0x3000000)Number              = 2230
    (0x1000000)RecoverableException = (
      (0x3000000)File              = 'F:\build\S000_D\src\DataFlowEngine\ImbRdl\ImbRdlExternalDb.cpp'
      (0x3000000)Line              = 278
      (0x3000000)Function          = 'SqlExternalDbStmt::executeStmt'
      (0x3000000)Type              = 'ComIbmComputeNode'
      (0x3000000)Name              = 'acd8f35d-e700-0000-0080-b78796c5e70d'
      (0x3000000)Label             = 'esql_13485_check_defect.Compute1'
      (0x3000000)Text              = 'The following error occurred execution SQL statement &3. inserts where &4'
      (0x3000000)Catalog           = 'WMQIv210'
      (0x3000000)Severity          = 3
      (0x3000000)Number            = 2519
      (0x1000000)Insert            = (
        (0x3000000)Type = 2
        (0x3000000)Text = '1'
      )
      (0x1000000)Insert            = (
        (0x3000000)Type = 2
        (0x3000000)Text = '1'
      )
      (0x1000000)Insert            = (
        (0x3000000)Type = 5
        (0x3000000)Text = 'USERDB'
      )
      (0x1000000)Insert            = (
        (0x3000000)Type = 5
        (0x3000000)Text = 'DELETE FROM DB2ADMIN.STOCK WHERE (STOCK_ID)=(?)'
      )
      (0x1000000)Insert            = (
        (0x3000000)Type = 5
        (0x3000000)Text = '500027,  '
      )
      (0x1000000)DatabaseException = (
        (0x3000000)File     = 'F:\build\S000_D\src\DataFlowEngine\ImbOdbc.cpp'
        (0x3000000)Line     = 153
        (0x3000000)Function = 'ImbOdbcHandle::checkRcInner'
        (0x3000000)Type     = ''
        (0x3000000)Name     = ''
        (0x3000000)Label    = ''
        (0x3000000)Text     = 'Root SQL exception'
        (0x3000000)Catalog  = 'WMQIv210'
        (0x3000000)Severity = 3
        (0x3000000)Number   = 2321
        (0x1000000)Insert   = (
          (0x3000000)Type = 2
          (0x3000000)Text = '100'
        )
      )
    )
  )
)
```

Conversion exception trace output

The following figure shows an extract of the output that might be generated by a Trace node that has its property *Pattern* set to a value that represents a structure that includes the ExceptionList tree.

The exception shown occurred when a conversion (CAST) exception was detected.

```
ExceptionList = (
  (0x1000000)RecoverableException = (
    (0x3000000)File                = 'F:\build\S000_D\src\DataFlowEngine\ImbComputeNode.cpp'
    (0x3000000)Line                = 402
    (0x3000000)Function            = 'ImbComputeNode::evaluate'
    (0x3000000)Type                = 'ComIbmComputeNode'
    (0x3000000)Name                = 'acd8f35d-e700-0000-0080-b78796c5e70d'
    (0x3000000)Label               = 'esql_13485_check_defect.Compute1'
    (0x3000000)Text                = 'Caught exception and rethrowing'
    (0x3000000)Catalog             = 'WMQIv210'
    (0x3000000)Severity            = 3
    (0x3000000)Number              = 2230
    (0x1000000)RecoverableException = (
      (0x3000000)File               = 'F:\build\S000_D\src\DataFlowEngine\ImbRdl\ImbRdlTypeCast.cpp'
      (0x3000000)Line               = 163
      (0x3000000)Function           = 'SqlTypeCast::evaluate'
      (0x3000000)Type               = ''
      (0x3000000)Name               = ''
      (0x3000000)Label              = ''
      (0x3000000)Text               = 'Error casting from %3 to %4'
      (0x3000000)Catalog            = 'WMQIv210'
      (0x3000000)Severity           = 3
      (0x3000000)Number             = 2521
      (0x1000000)Insert             = (
        (0x3000000)Type = 2
        (0x3000000)Text = '12'
      )
      (0x1000000)Insert             = (
        (0x3000000)Type = 2
        (0x3000000)Text = '28'
      )
      (0x1000000)Insert             = (
        (0x3000000)Type = 5
        (0x3000000)Text = 'CHARACTER'
      )
      (0x1000000)Insert             = (
        (0x3000000)Type = 5
        (0x3000000)Text = 'INTEGER'
      )
      (0x1000000)ConversionException = (
        (0x3000000)File     = 'F:\build\S000_D\src\CommonServices\ImbUtility.cpp'
        (0x3000000)Line     = 195
        (0x3000000)Function = 'imbWcsToInt64'
        (0x3000000)Type     = ''
        (0x3000000)Name     = ''
        (0x3000000)Label    = ''
        (0x3000000)Text     = 'Invalid characters'
        (0x3000000)Catalog  = 'WMQIv210'
        (0x3000000)Severity = 3
        (0x3000000)Number   = 2595
        (0x1000000)Insert   = (
          (0x3000000)Type = 5
          (0x3000000)Text = 'fred'
        )
      )
    )
  )
)
```

Parser exception trace output

The following figure shows an extract of the output that might be generated by a Trace node that has its property *Pattern* set to a value that represents a structure that includes the ExceptionList tree.

The exception shown occurred when the message parser was invoked.

```
ExceptionList = (
  (0x1000000)RecoverableException = (
    (0x3000000)File            = 'F:\build\S000_D\src\DataFlowEngine\ImbMqOutputNode.cpp'
    (0x3000000)Line            = 1444
    (0x3000000)Function        = 'ImbMqOutputNode::evaluate'
    (0x3000000)Type            = 'ComIbmMQOutputNode'
    (0x3000000)Name            = 'c76eb6cd-e600-0000-0080-b78796c5e70d'
    (0x3000000)Label           = 'esql_13485_check_defect.OUT'
    (0x3000000)Text            = 'Caught exception and rethrowing'
    (0x3000000)Catalog         = 'WMQIv210'
    (0x3000000)Severity        = 3
    (0x3000000)Number          = 2230
    (0x1000000)ParserException = (
      (0x3000000)File            = 'F:\build\S000_D\src\MTI\MTIforBroker\GenXmlParser2\XmlImbParser.cpp'
      (0x3000000)Line            = 210
      (0x3000000)Function        = 'XmlImbParser::refreshBitStreamFromElements'
      (0x3000000)Type            = 'ComIbmMQInputNode'
      (0x3000000)Name            = 'ce64b6cd-e600-0000-0080-b78796c5e70d'
      (0x3000000)Label           = 'esql_13485_check_defect.IN'
      (0x3000000)Text            = 'XML Writing Errors have occurred'
      (0x3000000)Catalog         = 'WMQIv210'
      (0x3000000)Severity        = 3
      (0x3000000)Number          = 5010
      (0x1000000)ParserException = (
        (0x3000000)File     = 'F:\build\S000_D\src\MTI\MTIforBroker\GenXmlParser2\XmlImbParser.cpp'
        (0x3000000)Line     = 551
        (0x3000000)Function = 'XmlImbParser::checkForBodyElement'
        (0x3000000)Type     = ''
        (0x3000000)Name     = ''
        (0x3000000)Label    = ''
        (0x3000000)Text     = 'No valid body of the document could be found.'
        (0x3000000)Catalog  = 'WMQIv210'
        (0x3000000)Severity = 3
        (0x3000000)Number   = 5005
      )
    )
  )
)
```

User exception trace output

The following figure shows an extract of the output that might be generated by a Trace node that has its property *Pattern* set to a value that represents a structure that includes the ExceptionList tree.

The exception shown occurred when a user exception was generated (with the ESQL THROW statement).

```
ExceptionList = (
  (0x1000000)RecoverableException = (
    (0x3000000)File          = 'F:\build\S000_D\src\DataFlowEngine\ImbComputeNode.cpp'
    (0x3000000)Line          = 402
    (0x3000000)Function      = 'ImbComputeNode::evaluate'
    (0x3000000)Type          = 'ComIbmComputeNode'
    (0x3000000)Name          = 'acd8f35d-e700-0000-0080-b78796c5e70d'
    (0x3000000)Label         = 'esql_13485_check_defect.Compute1'
    (0x3000000)Text          = 'Caught exception and rethrowing'
    (0x3000000)Catalog       = 'WMQIv210'
    (0x3000000)Severity      = 3
    (0x3000000)Number        = 2230
    (0x1000000)UserException = (
      (0x3000000)File     = 'F:\build\S000_D\src\DataFlowEngine\ImbRdl\ImbRdlThrowExceptionStatements.cpp'
      (0x3000000)Line     = 148
      (0x3000000)Function = 'SqlThrowExceptionStatement::execute'
      (0x3000000)Type     = 'ComIbmComputeNode'
      (0x3000000)Name     = 'acd8f35d-e700-0000-0080-b78796c5e70d'
      (0x3000000)Label    = 'esql_13485_check_defect.Compute1'
      (0x3000000)Text     = 'User Generated SQL 'USER' exception'
      (0x3000000)Catalog  = 'WMQIv210'
      (0x3000000)Severity = 1
      (0x3000000)Number   = 2949
      (0x1000000)Insert   = (
        (0x3000000)Type = 5
        (0x3000000)Text = 'USER'
      )
      (0x1000000)Insert   = (
        (0x3000000)Type = 5
        (0x3000000)Text = 'Insert1'
      )
      (0x1000000)Insert   = (
        (0x3000000)Type = 5
        (0x3000000)Text = 'Insert2'
      )
      (0x1000000)Insert   = (
        (0x3000000)Type = 5
        (0x3000000)Text = 'etc'
      )
      (0x1000000)Insert   = (
        (0x3000000)Type = 5
        (0x3000000)Text = ''
      )
      (0x1000000)Insert   = (
        (0x3000000)Type = 5
        (0x3000000)Text = ''
      )
      (0x1000000)Insert   = (
        (0x3000000)Type = 5
        (0x3000000)Text = ''
      )
      (0x1000000)Insert   = (
        (0x3000000)Type = 5
        (0x3000000)Text = ''
      )
      (0x1000000)Insert   = (
        (0x3000000)Type = 5
        (0x3000000)Text = ''
      )
    )
  )
)
```

Message flow porting

If you have configured a message flow that runs on a broker on a distributed system, and you now want to deploy it to a broker that runs on z/OS, you might have to take additional actions to port the flow successfully.

Consider the following resources and attributes:

WebSphere MQ queue manager and queue names
WebSphere MQ imposes some restrictions for resource names on z/OS:

- The queue manager name cannot be greater than four characters.
- All queue names must be in uppercase. Although using quotation marks preserves the case, certain WebSphere MQ activities on z/OS cannot find the queue names being referenced.

For more information about configuring on z/OS, refer to the *Concepts and Planning Guide* section of the WebSphere MQ Version 7 Information Center online or WebSphere MQ Version 6 Information Center online

File system references
File system references must reflect a UNIX file path. If you deploy a message flow to z/OS that you have previously run on Windows, you might have to make changes. If you have previously deployed the message flow to a UNIX system (AIX, Linux, Solaris, or HP-UX), you do not have to make any changes.

Databases
If the message flow accesses one or more databases, it might be subject to some restrictions based on the system on which the database is defined. These restrictions are described in Database locations.

Monitoring message flows

The following reference topics on monitoring message flows are available:

- "Monitoring profile"
- "The monitoring event" on page 1495
- "Correlation and monitoring events" on page 1497

Monitoring profile

To customize events after a message flow has been deployed, but without redeploying the flow, use a monitoring profile configurable service. By using this service, you can apply a monitoring profile to one or more message flows.

A monitoring profile is an XML document that specifies the event sources in a message flow that will emit events, and the properties of those events. The monitoring profile XML must conform to XML schema file `MonitoringProfile.xsd`.

Outline monitoring profile

Here is an outline monitoring profile that contains a single event source to illustrate the structure:

```
<p:monitoringProfile
xmlns:p="http://www.ibm.com/xmlns/prod/websphere/messagebroker/6.1.0/monitoring/profile">
    <p:eventSource p:enabled="true" p:eventSourceAddress="SOAPInput.transaction.Start">
        <p:eventPointDataQuery>
            <p:eventIdentity>
                <p:eventName p:literal="" p:queryText=""/>
            </p:eventIdentity>
```

```
            <p:eventCorrelation>
                <p:localTransactionId p:queryText="" p:sourceOfId="automatic"/>
                <p:parentTransactionId p:queryText="" p:sourceOfId="automatic"/>
                <p:globalTransactionId p:queryText="" p:sourceOfId="automatic"/>
            </p:eventCorrelation>
            <p:eventSequence p:name="creationTime"/>        </p:eventPointDataQuery>
        <p:applicationDataQuery>
            <p:simpleContent p:dataType="boolean" p:name="" p:targetNamespace="">
                <p:valueQuery p:queryText=""/>
            </p:simpleContent>
            <p:complexContent p:name="" targetNamespace="">
                <p:payloadQuery p:queryText=""/>
            </p:complexContent>
        </p:applicationDataQuery>
        <p:bitstreamDataQuery p:bitstreamContent="all" p:encoding="base64Binary"/>
    </p:eventSource>
</p:monitoringProfile>
```

The root element is `p:monitoringProfile`. It contains one or more `p:eventSource` elements, each of which specifies an event source and defines its properties. Each `p:eventSource` element contains:

- A `p:eventPointDataQuery` element that provides key information about the event.
- Optional: A `p:applicationDataQuery` element if the event payload will include data fields extracted from a message.
- Optional: A `p:bitstreamDataQuery` element if the event payload will include bitstream data from a message.

Creating a monitoring profile

To help you create monitoring profiles, the following sample contains an outline monitoring profile XML file and the monitoring profile XML schema file (`MonitoringProfile.xsd`):

- WebSphere Business Monitor

Validate your monitoring profiles against the XML schema to ensure that they are correct.

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

Tip: If you have a deployed message flow that has monitoring properties configured using the Message Flow editor, you can use the mqsireportflowmonitoring command to create the equivalent monitoring profile XML file for the message flow. You can use this profile as a starting point for creating other monitoring profiles.

The following steps describe how to create a monitoring profile XML. Follow these steps for each `p:eventSource` element.

1. Specify the `p:eventSource/@p:eventSourceAddress` attribute.

   This is a string that uniquely identifies the event source in the message flow. It must follow the fixed format for transaction events and terminal events, as shown in the following table:

Event type	Event source address
Transaction Start event	*nodelabel*.transaction.Start
Transaction End event	*nodelabel*.transaction.End
Transaction Rollback event	*nodelabel*.transaction.Rollback

Event type	Event source address
Terminal event	*nodelabel*.terminal.<Terminal>

Note: *nodelabel* is the label of the node as known by the broker runtime components. If the node is in a subflow the label reflects this. For example, flow A contains an instance of flow B as a subflow labeled `myB`; flow B contains an instance of a Compute node labeled `myCompute`. The *nodelabel* for the Compute node is `myB.myCompute`.

In the emitted event, the address string of the event source is set in the `wmb:eventData/@wmb:eventSourceAddress` attribute.

2. Optional: Specify the name by which events from this event source will be known, in the `p:eventPointDataQuery/p:eventIdentity/p:eventName` element.
   - If the event name is a fixed string, complete the `p:eventName/@p:literal` attribute
   - If the event name is to be extracted from a field in the message, complete the `p:eventName/@p:queryText` attribute by specifying an XPath query.

   In the emitted event, the event name is set in the `wmb:eventPointData/wmb:eventIdentity/@wmb:eventName` attribute.

   If `p:eventName` element is not supplied, `@wmb:eventName` in the emitted event defaults to @p:eventSourceAddress.
3. Optional: Complete the `p:applicationDataQuery` element, if the event is to contain selected data fields extracted from the message. You can extract one or more fields from the message data and include it with the event. The fields can be simple or complex.
   - For each simple data field, complete a `p:simpleContent` element:
     - Complete the `p:simpleContent/p:valueQuery/@p:queryText` attribute by specifying an XPath query.
     - Complete the `p:simpleContent/@p:name`, `@p:namespace` and `@p:dataType` attributes. The `@p:dataType` value must be one of boolean, date, dateTime, decimal, duration, integer, string or time.
   - For each complex data field, complete a `p:complexContent` element:
     - Complete the `p:complexContent/p:payloadQuery/@p:queryText` attribute by specifying an XPath query.
     - Complete the `p:complexContent/@p:name` and `@p:namespace` attributes.

   This facility is commonly used for communicating significant business data in a business event. If the event contains the input bit stream, this facility can also be used to extract key fields, allowing another application to provide an audit trail or to resubmit failed messages.

   In the emitted event, the extracted data is set in the `wmb:applicationData/wmb:simpleContent` and `wmb:applicationData/wmb:complexContent` elements.
4. Optional: Complete the `p:bitstreamDataQuery` element, if the event is to capture message bitstream data:
   - Complete the `@p:bitstreamContent` attribute. The attribute value must be one of headers, body, or all.
   - Complete the `@p:encoding` attribute. The attribute value must be one of CDATA, base64Binary or hexBinary.

   In the emitted event, the extracted bitstream data is set in the `wmb:bitstreamData/wmb:bitstream` element.

5. Optional: Complete the p:eventPointDataQuery/p:eventCorrelation element. For information about correlation, see "Correlation and monitoring events" on page 1497.

   Every emitted monitoring event must contain at least one correlation attribute, and can contain up to three. If no correlation information is specified in the monitoring profile, the first event source in the message flow generates a unique identifier for the local correlator, and saves it in the Environment tree. All later event sources in the message flow use this local correlator value.

   a. Optional: Complete the p:localTransactionId element.
      - If you want to reuse the local correlator from the Environment tree, set the p:localTransactionId/@p:sourceOfId attribute to automatic. If no local correlator exists yet, a new unique value will be generated and saved in the Environment tree.
      - If you want to use a value contained in a location in the message, set the p:localTransactionId/@p:sourceOfId attribute to query, then complete the p:localTransactionId/@p:queryText attribute by specifying an XPath query. Ensure that the specified location contains a correlator value unique to this invocation of the message flow. The value is saved in the Environment tree.

   b. Optional: Complete the p:parentTransactionId element.
      - If you want to reuse the parent correlator from the Environment tree, set the p:parentTransactionId/@p:sourceOfId attribute to automatic. If no parent correlator exists yet, no parent correlator will be used.
      - If you want to use a value contained in a location in the message, set the p:parentTransactionId/@p:sourceOfId attribute to query, then complete the p:parentTransactionId/@p:queryText attribute by specifying an XPath query. Ensure that the specified location contains a suitable value for the parent correlator. The value is saved in the Environment tree.

   c. Optional: Complete the p:globalTransactionId element.
      - If you want to reuse the global correlator from the Environment tree, set the p:globalTransactionId/@p:sourceOfId attribute to automatic. If no global correlator exists yet, no global correlator will be used.
      - If you want to use a value contained in a location in the message, set the p:globalTransactionId/@p:sourceOfId attribute to query, then complete the p:globalTransactionId/@p:queryText attribute by specifying an XPath query. Ensure that the specified location contains a suitable value for the global correlator. The value is saved in the Environment tree.

XPath queries and XML namespaces

If an XPath query contains a component that has an XML namespace, the XPath contains a namespace prefix for the namespace. For example, the following XPath refers to components in two different namespaces:

```
<p:localTransactionId p:sourceOfId="query" p:queryText="$Body/soapenv:Header/wsa:messageID" />
```

For the broker to resolve the namespace prefix, the namespace URL must also be provided. Supply a prefix mapping element for each namespace:

```
<p:localTransactionId p:sourceOfId="query" p:queryText="$Body/soapenv:Header/wsa:messageID">
  <p:prefixMapping p:prefix="soapenv" p:URI="http://www.w3.org/2003/05/soap-envelope" />
  <p:prefixMapping p:prefix="wsa" p:URI="http://www.w3.org/2005/08/addressing" />
</p:localTransactionId>
```

Monitoring profile examples

The following XML documents conform to the monitoring profile schema

Monitoring profile 1: Two event sources, each supplying an event name

```
<p:monitoringProfile p:version="2.0">
 <p:eventSource p:eventSourceAddress="SOAPInput.transaction.Start">
  <p:eventPointDataQuery>
    <p:eventIdentity>
      <p:eventName p:literal="SOAP start event"/>
    </p:eventIdentity>
  </p:eventPointDataQuery>
 </p:eventSource>
 <p:eventSource p:eventSourceAddress="SOAPInput.transaction.End">
  <p:eventPointDataQuery>
    <p:eventIdentity>
      <p:eventName p:literal="SOAP end event"/>
    </p:eventIdentity>
  </p:eventPointDataQuery>
 </p:eventSource>
</p:monitoringProfile>
```

Monitoring profile 2: Supply an alternative local correlator

```
<p:monitoringProfile p:version="2.0">
 <p:eventSource p:eventSourceAddress="SOAPInput.transaction.Start">
  <p:eventPointDataQuery>
    <p:eventCorrelation>
      <p:localTransactionId p:queryText="$Body/soapenv:Header/wsa:messageID" p:sourceOfId="query">
        <p:prefixMapping p:prefix="soapenv" p:URI="http://www.w3.org/2003/05/soap-envelope"/>
        <p:prefixMapping p:prefix="wsa" p:URI="http://www.w3.org/2005/08/addressing"/>
      </p:localTransactionId>
    </p:eventCorrelation>
  </p:eventPointDataQuery>
 </p:eventSource>
</p:monitoringProfile>
```

Monitoring profile 3: Include two simple fields from the message

```
<p:monitoringProfile p:version="2.0">
 <p:eventSource p:eventSourceAddress="MQInput.terminal.out">
  <p:applicationDataQuery>
    <p:simpleContent p:dataType="integer" p:name="InvoiceNumber">
      <p:valueQuery p:queryText="$Body/invoice/invoiceNo"/>
    </p:simpleContent>
    <p:simpleContent p:dataType="string" p:name="BatchID">
      <p:valueQuery p:queryText="$Body/batch/batchNo"/>
    </p:simpleContent>
  </p:applicationDataQuery>
 </p:eventSource>
</p:monitoringProfile>
```

Monitoring profile 4: Include the bitstream, encoded as CDATA

By default, bitstreams are encoded in base64Binary format. The following monitoring profile changes the encoding to CDATA.

```
<p:monitoringProfile p:version="2.0">
 <p:eventSource p:eventSourceAddress="MQInput.terminal.out">
    <p:bitstreamDataQuery p:bitstreamContent="body" p:encoding="CDATA"/>
 </p:eventSource>
</p:monitoringProfile>
```

CDATA encoding is not suitable for all types of data. Use CDATA only when `@p:bitstreamContent="body"`. Do not use CDATA if your message bitstreams might contain characters that are not allowed in XML (see http://www.w3.org/TR/2006/REC-xml-20060816/#charsets).

The monitoring event

You can configure WebSphere Message Broker to emit a monitoring event (an XML document) when something interesting happens. Events are typically emitted to support transaction monitoring, transaction auditing, and business process monitoring. The event XML conforms to the monitoring event schema WMBEvent.xsd.

Each event contains the following information:

- Source of the event
- Name of the event
- Creation time
- Correlation ID for events emitted by the same transaction or unit of work
- Details of the message flow

Tip: A terminal emits an event only if the message passes through that terminal. In particular, the output terminal of an output node emits events only if it is connected.

A monitoring event can also contain the following items:

- Application data extracted from the message.
- Part or all of the message bit stream.

  The bit stream cannot be included in monitoring events if one of the following nodes created the message:

  – FileInput, if the message domain is MRM or XMLNSC
  – TCPIPClientInput, if the message domain is MRM or XMLNSC
  – TCPIPServerInput, if the message domain is MRM or XMLNSC
  – SOAPInput, SOAPAsyncResponse
  – Any of the Adapter input nodes (PeopleSoftInput, SAPInput, SiebelInput, or TwineballInput)

  The bit stream cannot be included in monitoring events if the message is a response from one of the following nodes:

  – SOAPRequest
  – Any of the Adapter request nodes (SAPRequest, PeopleSoftRequest, SiebelRequest, TwineballRequest)

  A warning is logged to user trace if an event source is configured to include the bit stream, but it cannot be included. You might be able to extract some or all of the data in the message as Event Payload; be aware that large event messages might affect performance.

Use either the monitoring properties of a message flow or a monitoring profile configurable service to configure the following items:

- Where an event is emitted from
- The content of the event

Example event

```
<?xml version="1.0" encoding="UTF-8"?>
<wmb:event xmlns:wmb=http://www.ibm.com/xmlns/prod/websphere/messagebroker/6.1.0/monitoring/event>
   <wmb:eventPointData>
     <wmb:eventData wmb:eventSourceAddress="MQInput1.terminal.in"
                    wmb:eventSchemaVersion="6.1.0.3" wmb:productVersion="6103">
       <wmb:eventIdentity wmb:eventName="MQInput event"/>
```

```
      <wmb:eventSequence wmb:creationTime="2001-12-31T12:00:00"/>
      <wmb:eventCorrelation wmb:localTransactionId="123"
                            wmb:parentTransactionId="456"
                            wmb:globalTransactionId="789"/>
    </wmb:eventData>
    <wmb:messageFlowData>
      <wmb:broker wmb:UUID="d53122ae-1c01-0000-0080-b1b02528c6bf"
                  wmb:name="myBroker"/>
      <wmb:executionGroup wmb:UUID="d43122ae-1c01-0000-0080-b1b02528c6bf"
                          wmb:name="default"/>
      <wmb:messageFlow wmb:UUID="e6d224ae-1c01-0000-0080-9100cd1a61f7"
                       wmb:name="myMessageFlow" wmb:threadId="4201"
                       wmb:uniqueFlowName="myBroker.default.myMessageFlow"/>
      <wmb:node wmb:nodeLabel="MQInput1" wmb:nodeType="ComIbmMqInputNode"
                wmb:terminal="in" wmb:detail="MYMESSAGEFLOW.IN"/>
    </wmb:messageFlowData>
  </wmb:eventPointData>
  <wmb:applicationData xmlns="">
    <wmb:simpleContent wmb:name="invoiceNo" wmb:targetNamespace=""
                       wmb:dataType="string" wmb:value="567"/>
    <wmb:complexContent wmb:elementName="customerName" wmb:targetNamespace="">
      <customerName>
        <firstName>Steve</firstName>
        <lastName>Bloggs</lastName>
      </customerName>
    </wmb:complexContent>
  </wmb:applicationData>
  <wmb:bitstreamData>
    <wmb:bitstream wmb:encoding="base64Binary">TUQgIAIAAAAAAAAACAAAAP////8AAAAAIgIAALUBAAAgICAgICAg
IAAAAAAAAAAAQU1RIFFNMSAgICAgICAgIHo640ggABsHAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAgICAgICAgICAgICAgI
CAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIFFNMSAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIC
Ag</wmb:bitstream>
  </wmb:bitstreamData>
</wmb:event>
```

Default content of monitoring events

When an event is emitted, the fields in the event are created using the information provided by the monitoring properties of the message flow, or a monitoring profile configurable service if one has been applied to the message flow. Any event field that is not explicitly specified is given a default value as shown in the table.

Field in event	Default Value
eventData/@wmb:eventSourceAddress	No default; you must provide this information.
eventData/@wmb:eventSchemaVersion	6.1.0.3
eventData/@wmb:productVersion	6103
eventData/eventIdentity/ @wmb:eventName	The default is derived from @eventSourceAddress
eventData/eventSequence/@creationTime	The date and time when the event was created.
eventData/eventCorrelation/ @localTransactionId	A generated unique identifier.
eventData/eventCorrelation/ @parentTransactionId	No default. Unless you set this value, an empty string is used.
eventData/eventCorrelation/ @globalTransactionId	No default. Unless you set this value, an empty string is used.
messageFlowData/broker/@name	The name of the broker.
messageFlowData/broker/@UUID	The UUID of the broker.

Field in event	Default Value
messageFlowData/executionGroup/@name	The name of the execution group.
messageFlowData/executionGroup/@UUID	The UUID of the execution group.
messageFlowData/messageFlow/@name	The name of the message flow.
messageFlowData/messageFlow/@UUID	The UUID of the message flow.
messageFlowData/messageFlow/@uniqueFlowName	A string composed of the names of the broker, execution group, and flow in the form: *brokerName.executionGroupName.flowName*
messageFlowData/messageFlow/@threadId	The thread ID of the message flow. The format depends on the platform.
messageFlowData/node/@nodeLabel	The label of the node that emitted the event.
messageFlowData/node/@nodeType	The type of the node that emitted the event.
messageFlowData/node/@nodeDetail	Optional information about the node. **MQInput** The name of the queue. **Other nodes** Omitted.
applicationData	No default; omitted if not provided.
bitstreamData	No default; omitted if not provided.

Event message persistence

To receive events that are published as persistent messages, register your subscription setting Pers in the **<RegOpt>** registration option in your register subscriber request; see Register Subscriber message.

Correlation and monitoring events

A monitoring application uses correlation attributes to identify events that belong to the same business transaction.

A business transaction can be any of the following scenarios:

- A single invocation of a message flow.
- Multiple invocations of the same message flow from a parent application. The parent application might be another message flow.
- Multiple invocations of various message flows from a parent application. The parent application might be another message flow.

Three correlation attributes are available for you to use in your events: *local correlator*, *parent correlator* and *global correlator*. The exact usage of the correlation attributes varies depending on the requirements. For example, a parent application can pass its transaction identifier to the child message flow (perhaps in a header) so that the child message flow can report it in the event as a parent correlator.

Every emitted monitoring event must contain at least a local correlator, and can also contain a parent correlator and global correlator. Correlation information is placed in the following attributes of the event:

```
wmb:eventPointData/wmb:eventCorrelation/@wmb:localTransactionId
wmb:eventPointData/wmb:eventCorrelation/@wmb:parentTransactionId
wmb:eventPointData/wmb:eventCorrelation/@wmb:globalTransactionId
```

You can specify correlation information when you configure the event.

If you do not specify any correlation information when you configure your events, the first event source in the message flow automatically allocates a local correlator. This is saved in the Environment tree, and is used by all later event sources in the same transaction. The parent correlator and global correlator are set to empty strings.

If you do specify correlation information, you must configure the correlation attributes to be used, and where they will read their value from. Typically you need to specify correlation information only for the first event source in the message flow; by default all later event sources retrieve the same value from the Environment tree.

The exact steps for specifying correlation information depend on whether you are using monitoring properties or a monitoring profile to configure your events, but the principle is the same for both techniques:

Local correlator

If you want to reuse the local correlator from the Environment tree, specify Automatic. If no local correlator exists yet, a new unique value will be generated and saved in the Environment tree.

If you want to use a value contained in a location in the message, specify the location of the correlator by supplying an XPath into the message tree. Ensure that the specified location contains a correlator value unique to this invocation of the message flow. The extracted value is saved in the Environment tree as the local correlator.

Parent correlator

If you want to reuse the parent correlator from the Environment tree, specify Automatic. If no parent correlator exists yet, no parent correlator will be used.

If you want to use a value contained in a location in the message, specify the location of the correlator by supplying an XPath into the message tree. Ensure that the specified location contains a suitable value for the parent correlator. The extracted value is saved in the Environment tree as the parent correlator.

Global correlator

If you want to reuse the global correlator from the Environment tree, specify Automatic. If no global correlator exists yet, no global correlator will be used.

If you want to use a value contained in a location in the message, specify the location of the correlator by supplying an XPath into the message tree. Ensure that the specified location contains a suitable value for the global correlator. The extracted value is saved in the Environment tree as the global correlator.

When a correlator value has been set, it is saved in the Environment tree. Later event sources can reuse the saved value by specifying Automatic. There is no need to use the same XPath in all the event sources in your message flow, and doing so might adversely affect performance.

The locations in the Environment tree used to save correlator values for use by later events are:

```
Environment.Monitoring.EventCorrelation.localTransactionId
Environment.Monitoring.EventCorrelation.parentTransactionId
Environment.Monitoring.EventCorrelation.globalTransactionId
```

The following message tree locations often contain a value that can be used as a correlator:

```
$Root/MQMD/MsgId
$Root/MQMD/CorrelId
$Root/JMSTransport/Transport_Folders/Header_Values/JMSMessageID
$Root/JMSTransport/Transport_Folders/Header_Values/JMSCorrelationID
$LocalEnvironment/Destination/HTTP/RequestIdentifier
$LocalEnvironment/Wildcard/WildcardMatch
```

Tip: If the three available correlators are not sufficient, you can configure the event to extract other correlation fields from the message and place them in the `wmb:applicationData/wmb:simpleContent` section of the event.

Tip: The Collector node and the AggregateControl node do not preserve the Environment tree for later nodes in the message flow. If you want to use the same correlator value later in the flow, ensure that the correlator value is available in the message tree, and that the first event source after the Collector or AggregateControl node specifies the location of the correlator by supplying an XPath into the message tree.

Scenarios

Scenario 1: In this scenario, all three correlators are used to monitor data that starts in an external process. Several message flows then transform the data.

- The `globalTransactionID` field contains an identifier from the message header or payload. This identifier correlates events from the external process and WebSphere Message Broker.
- The `parentTransactionID` correlates events in WebSphere Message Broker from different message flows.
- The `localTransactionID` correlates events from the same message flow.

Scenario 2: In this scenario, the `parentTransactionID` field is used to correlate request and reply messages between two message flows:

- The Request flow sends a **purchaseOrder** request to an external application for processing.
- The Reply flow receives a confirmation reply from the external application when the **purchaseOrder** has been processed.

You need to correlate the request and replies belonging to the same purchase order. You can do this by setting the `parentTransactionID` to a field in the **purchaseOrder**, such as a **purchaseOrderID**, which is available in both the request and reply.

Message flow accounting and statistics data

You can collect message flow accounting and statistics data in various output formats.

Details of the information that is collected, and the output formats in which it can be recorded, are provided in the following topics:
- Statistics details
- Data formats
- Example output

You can also find information on how to use accounting and statistics data to improve the performance of a message flow in this developerWorks article on message flow performance.

Message flow accounting and statistics details

You can collect message flow, thread, node, and terminal statistics for message flows.

Message flow statistics

One record is created for each message flow in an execution group. Each record contains the following details:
- Message flow name and UUID
- Execution group name and UUID
- Broker name and UUID
- Start and end times for data collection
- Type of data collected (snapshot or archive)
- Processor and elapsed time spent processing messages
- Processor and elapsed time spent waiting for input
- Number of messages processed
- Minimum, maximum, and average message sizes
- Number of threads available and maximum assigned at any time
- Number of messages committed and backed out
- Accounting origin

Thread statistics

One record is created for each thread assigned to the message flow. Each record contains the following details:
- Thread number (this has no significance and is for identification only)
- Processor and elapsed time spent processing messages
- Processor and elapsed time spent waiting for input
- Number of messages processed
- Minimum, maximum, and average message sizes

Node statistics

One record is created for each node in the message flow. Each record contains the following details:
- Node name
- Node type (for example MQInput)
- Processor time spent processing messages
- Elapsed time spent processing messages
- Number of times that the node is invoked
- Number of messages processed
- Minimum, maximum, and average message sizes

Terminal statistics

One record is created for each terminal on a node. Each record contains the following details:

- Terminal name
- Terminal type (input or output)
- Number of times that a message is propagated to this terminal

For further details about specific output formats, see the following topics:

- "User trace entries for message flow accounting and statistics data" on page 1505
- "XML publication for message flow accounting and statistics data"
- "z/OS SMF records for message flow accounting and statistics data" on page 1508

Message flow accounting and statistics output formats

The message flow accounting and statistics data can be written in three formats.

For more information about the available output formats, see the following topics:

- User trace entries
- XML publication
- z/OS SMF records

XML publication for message flow accounting and statistics data

Certain information is written to the XML publication for message flow accounting and statistics data.

The data is created in the folder `WMQIStatisticsAccounting`, which contains sub-folders that provide more detailed information. All folders are present in the publication even if you set current data collection parameters to specify that the relevant data is not collected.

Snapshot data is used for performance analysis, and is published as retained and non-persistent. Archive data is used for accounting where an audit trail might be required, and is published as retained and persistent. All publications are global and can be collected by a subscriber that has registered anywhere in the network. They can also be collected by more than one subscriber.

One XML publication is generated for each message flow that is producing data for the time period that you have chosen. For example, if MessageFlowA and MessageFlowB are both producing archive data over a period of 60 minutes, both MessageFlowA and MessageFlowB produce an XML publication every 60 minutes.

If you are concerned about the safe delivery of these messages (for example, for charging purposes), use a secure delivery mechanism such as WebSphere MQ.

The folders and sub-folders in the XML publication have the following identifiers:

- WMQIStatisticsAccounting
- MessageFlow
- Threads
- ThreadStatistics
- Nodes
- NodesStatistics
- TerminalStatistics

The tables provided here describe the contents of each of these folders.

The following table describes the general accounting and statistics information, created in folder WMQIStatisticsAccounting.

Field	Data type	Details
RecordType	Character	Type of output, one of: • Archive • Snapshot
RecordCode	Character	Reason for output, one of: • MajorInterval • Snapshot • Shutdown • ReDeploy • StatsSettingsModified

The following table describes the message flow statistics information, created in folder MessageFlow.

Field	Data type	Details
BrokerLabel	Character (maximum 32)	Broker name
BrokerUUID	Character (maximum 32)	Broker universal unique identifier
ExecutionGroupName	Character (maximum 32)	Execution group name
ExecutionGroupUUID	Character (maximum 32)	Execution group universal unique identifier
MessageFlowName	Character (maximum 32)	Message flow name
StartDate	Character	Interval start date (YYYY-MM-DD)
StartTime	Character	Interval start time (HH:MM:SS:NNNNNN)
EndDate	Character	Interval end date (YYYY-MM-DD)
EndTime	Character	Interval end time (HH:MM:SS:NNNNNN)
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
MaximumElapsedTime	Numeric	Maximum elapsed time spent processing an input message (microseconds)
MinimumElapsedTime	Numeric	Minimum elapsed time spent processing an input message (microseconds)
TotalCPUTime	Numeric	Total processor time spent processing input messages (microseconds)
MaximumCPUTime	Numeric	Maximum processor time spent processing an input message (microseconds)
MinimumCPUTime	Numeric	Minimum processor time spent processing an input message (microseconds)

Field	Data type	Details
CPUTimeWaitingForInputMessage	Numeric	Total processor time spent waiting for input messages (microseconds)
ElapsedTimeWaitingForInputMessage	Numeric	Total elapsed time spent waiting for input messages (microseconds)
TotalInputMessages	Numeric	Total number of messages processed
TotalSizeOfInputMessages	Numeric	Total size of input messages (bytes)
MaximumSizeOfInputMessages	Numeric	Maximum input message size (bytes)
MinimumSizeOfInputMessages	Numeric	Minimum message input size (bytes)
NumberOfThreadsInPool	Numeric	Number of threads in pool
TimesMaximumNumberofThreadsReached	Numeric	Number of times the maximum number of threads is reached
TotalNumberOfMQErrors[1]	Numeric	Number of MQGET errors (MQInput node) or Web services errors (HTTPInput node)
TotalNumberOfMessagesWithErrors[2]	Numeric	Number of messages that contain errors
TotalNumberOfErrorsProcessingMessages	Numeric	Number of errors processing a message
TotalNumberOfTimeOutsWaitingForRepliesToAggregateMessages	Numeric	Number of timeouts processing a message (AggregateReply node only)
TotalNumberOfCommits	Numeric	Number of transaction commits
TotalNumberOfBackouts	Numeric	Number of transaction backouts
AccountingOrigin	Character (maximum 32)	Accounting origin

Notes:
1. For example, a conversion error occurs when the message is got from the queue.
2. These errors include exceptions that are thrown downstream of the input node, and errors detected by the input node after it has successfully retrieved the message from the queue but before it has propagated it to the out terminal (for example, a format error).

The following table describes the thread statistics information, created in folder Threads.

Field	Data type	Details
Number	Numeric	Number of thread statistics sub-folders in Threads folder

The following table describes the thread statistics information for each individual thread, created in folder ThreadStatistics, a subfolder of Threads.

Field	Data type	Details
Number	Numeric	Relative thread number in pool
TotalNumberOfInputMessages	Numeric	Total number of messages processed by thread
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
TotalCUPTime	Numeric	Total processor time spent processing input messages (microseconds)
CPUTimeWaitingForInputMessage	Numeric	Total processor time spent waiting for input messages (microseconds)

Field	Data type	Details
ElapsedTimeWaitingForInputMessage	Numeric	Total elapsed time spent waiting for input messages (microseconds)
TotalSizeOfInputMessages	Numeric	Total size of input messages (bytes)
MaximumSizeOfInputMessages	Numeric	Maximum size of input messages (bytes)
MinimumSizeOfInputMessages	Numeric	Minimum size of input messages (bytes)

The following table describes the node statistics information, created in folder Nodes.

Field	Data type	Details
Number	Numeric	Number of node statistics sub-folders in Nodes folder

The following table describes the node statistics information for each individual node, created in folder NodesStatistics, a subfolder of Nodes.

Field	Data type	Details
Label	Character	Name of node (Label)
Type	Character	Type of node
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
MaximumElapsedTime	Numeric	Maximum elapsed time spent processing input messages (microseconds)
MinimumElapsedTime	Numeric	Minimum elapsed time spent processing input messages (microseconds)
TotalCPUTime	Numeric	Total processor time spent processing input messages (microseconds)
MaximumCPUTime	Numeric	Maximum processor time spent processing input messages (microseconds)
MinimumCPUTime	Numeric	Minimum processor time spent processing input messages (microseconds)
CountOfInvocations	Numeric	Total number of messages processed by this node
NumberOfInputTerminals	Numeric	Number of input terminals
NumberOfOutputTerminals	Numeric	Number of output terminals

The following table describes the terminal statistics information, created in folder TerminalStatistics.

Field	Data type	Details
Label	Character	Name of terminal
Type	Character	Type of terminal, one of: • Input • Output
CountOfInvocations	Numeric	Total number of invocations

User trace entries for message flow accounting and statistics data

Certain information is written to the user trace log for message flow accounting and statistics data.

The data records are identified by the following message numbers:
- BIP2380I
- BIP2381I
- BIP2382I
- BIP2383I

The inserts for each message are described in the following tables.

This table describes the inserts in message BIP2380I. One message is written for the message flow.

Field	Data type	Details
ProcessID	Numeric	Process ID
Key	Numeric	Key used to associate related accounting and statistics BIP messages
Type	Character	Type of output, one of: • Archive • Snapshot
Reason	Character	Reason for output, one of: • MajorInterval • Snapshot • Shutdown • ReDeploy • StatsSettingsModified
BrokerLabel	Character (maximum 32)	Broker name
BrokerUUID	Character (maximum 32)	Broker universal unique identifier
ExecutionGroupName	Character (maximum 32)	Execution group name
ExecutionGroupUUID	Character (maximum 32)	Execution group universal unique identifier
MessageFlowName	Character (maximum 32)	Message flow name
StartDate	Character	Interval start date (YYYY-MM-DD)
StartTime	Character	Interval start time (HH:MM:SS:NNNNNN)
EndDate	Character	Interval end date (YYYY-MM-DD)
EndTime	Character	Interval end time (HH:MM:SS:NNNNNN)
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)

Field	Data type	Details
MaximumElapsedTime	Numeric	Maximum elapsed time spent processing an input message (microseconds)
MinimumElapsedTime	Numeric	Minimum elapsed time spent processing an input message (microseconds)
TotalCPUTime	Numeric	Total processor time spent processing input messages (microseconds)
MaximumCPUTime	Numeric	Maximum processor time spent processing an input message (microseconds)
MinimumCPUTime	Numeric	Minimum processor time spent processing an input message (microseconds)
CPUTimeWaitingForInputMessage	Numeric	Total processor time spent waiting for input messages (microseconds)
ElapsedTimeWaitingForInputMessage	Numeric	Total elapsed time spent waiting for input messages (microseconds)
TotalInputMessages	Numeric	Total number of messages processed
TotalSizeOfInputMessages	Numeric	Total size of input messages (bytes)
MaximumSizeOfInputMessages	Numeric	Maximum input message size (bytes)
MinimumSizeOfInputMessages	Numeric	Minimum input message size (bytes)
NumberOfThreadsInPool	Numeric	Number of threads in pool
TimesMaximumNumberofThreadsReached	Numeric	Number of times the maximum number of threads is reached
TotalNumberOfMQErrors[1]	Numeric	Number of MQGET errors (MQInput node) or Web services errors (HTTPInput node)
TotalNumberOfMessagesWithErrors[2]	Numeric	Number of messages that contain errors
TotalNumberOfErrorsProcessingMessages	Numeric	Number of errors processing a message
TotalNumberOfTimeOutsWaitingForRepliesToAggregateMessages	Numeric	Number of timeouts processing a message (AggregateReply node only)
TotalNumberOfCommits	Numeric	Number of transaction commits
TotalNumberOfBackouts	Numeric	Number of transaction backouts
AccountingOrigin	Character (maximum 32)	Accounting origin

Notes:
1. For example, a conversion error occurs when the message is got from the queue.
2. These errors include exceptions that are thrown downstream of the input node, and errors that are detected by the input node after it has successfully retrieved the message from the queue (for example, a format error).

The following table describes the inserts in message BIP2381I. One message is written for each thread.

Field	Data type	Details
ProcessID	Numeric	Process ID
Key	Numeric	Key used to associate related accounting and statistics BIP messages
Number	Numeric	Relative thread number in pool
TotalNumberOfInputMessages	Numeric	Total number of messages processed by thread
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
TotalCUPTime	Numeric	Total processor time spent processing input messages (microseconds)
CPUTimeWaitingForInputMessage	Numeric	Total processor time spent waiting for input messages (microseconds)
ElapsedTimeWaitingForInputMessage	Numeric	Total elapsed time spent waiting for input messages (microseconds)
TotalSizeOfInputMessages	Numeric	Total size of input messages (bytes)
MaximumSizeOfInputMessages	Numeric	Maximum size of input messages (bytes)
MinimumSizeOfInputMessages	Numeric	Minimum size of input messages (bytes)

The following table describes the inserts in message BIP2382I. One message is written for each node.

Field	Data type	Details
ProcessID	Numeric	Process ID
Key	Numeric	Key used to associate related accounting and statistics BIP messages
Label	Character	Name of node (Label)
Type	Character	Type of node
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
MaximumElapsedTime	Numeric	Maximum elapsed time spent processing input messages (microseconds)
MinimumElapsedTime	Numeric	Minimum elapsed time spent processing input messages (microseconds)
TotalCPUTime	Numeric	Total processor time spent processing input messages (microseconds)
MaximumCPUTime	Numeric	Maximum processor time spent processing input messages (microseconds)
MinimumCPUTime	Numeric	Minimum processor time spent processing input messages (microseconds)
CountOfInvocations	Numeric	Total number of messages processed by this node
NumberOfInputTerminals	Numeric	Number of input terminals
NumberOfOutputTerminals	Numeric	Number of output terminals

The following table describes the inserts in message BIP2383I. One message is written for each terminal on each node.

Field	Data type	Details
ProcessID	Numeric	Process ID
Key	Numeric	Key used to associate related accounting and statistics BIP messages
Label	Character	Name of terminal
Type	Character	Type of terminal, one of: • Input • Output
CountOfInvocations	Numeric	Total number of invocations

z/OS SMF records for message flow accounting and statistics data

Certain information is written to z/OS SMF records for message flow accounting and statistics data.

The data records are type 117 records with the following identifiers:
- BipSMFDate
- BipSMFRecordHdr
- BipSMFTriplet
- BipSMFMessageFlow
- BipSMFThread
- BipSMFNode
- BipSMFTerminal

The following tables describe the contents of each of these records.

The following table describes the contents of the BipSMFDate record.

Field	Data type	Details
YYYY	signed short int	2 byte year
MM	char	1 byte month
DD	char	1 byte day

The following table describes the contents of the BipSMFRecordHdr record.

Field	Data type	Details
SM117LEN	unsigned short int	SMF record length
SM117SEG	unsigned short int	System reserved
SM117FLG	char	System indicator
SM117RTY	char	Record type 117 (x'75')
SM117TME	unsigned int	Time when SMF moved the record (time since midnight in hundredths of a second)
SM117DTE	unsigned int	Date when SMF moved the record in packed decimal form 0cyydddF where: c is 0 (19xx) or 1 (20xx) yy is the current year (0-99) ddd is the current day (1-366) F is the sign
SM117SID	unsigned int	System ID

Field	Data type	Details
SM117SSI	unsigned int	Subsystem ID
SM117STY	unsigned short int	Record subtype, one of : • 1 (only message flow or threads data is being collected) • 2 (node data is being collected)[1]
SM117TCT	unsigned int	Count of triplets
SM117SRT	unsigned char	Record type, one of: • Archive • Snapshot
SM117SRC	unsigned char	Record code, one of: • 00 = None • 01 = Major Interval • 02 = Snapshot • 03 = Shutdown • 04 = Redeploy • 05 = Stats Settings Modified
SM117RSQ	unsigned short int	Sequence number of the record when multiple records are written for a collection interval.
SM117NOR	unsigned short int	Total number of related records in a collection interval.

Note:

1. When only nodes data is being collected, a single subtype 2 record is written. If nodes and terminals data is being collected, multiple subtype 2 records are written.

The following table describes the contents of the BipSMFTriplet record.

Field	Data type	Details
TRPLTOSE	signed int	Offset of record from start of SMF record
TRPLTDLE	signed short int	Length of data type
TRPLTNDR	signed short int	Number of data types in SMF record

The following table describes the contents of the BipSMFMessageFlow record.

Field	Data type	Details
IMFLID	short int	Control block hex ID (BipSMFMessageFlow_ID)
IMFLLEN	short int	Length of control block
IMFLEYE	char[4]	Eyecatcher (IMFL)
IMFLVER	int	Version number (BipSMFRecordVersion)
IMFLBKNM	char32]	Broker name
IMFLBKID	char[36]	Broker universal unique identifier
IMFLEXNM	char[32]	Execution group name
IMFLEXID	char[36]	Execution group universal unique identifier
IMFLMFNM	char[32]	Message flow name
IMFLSTDT	BipSMFDate	Interval start date
IMFLSTTM	unsigned int	Interval start time (format as for SM117TME)
IMFLENDT	BipSMFDate	Interval end date
IMFLENTM	unsigned int	Interval end time (format as for SM117TME)

Field	Data type	Details
IMFLTPTM	long long int	Total elapsed time spent processing input messages (8 bytes binary, microseconds)
IMFLMXTM	long long int	Maximum elapsed time spent processing an input message (8 bytes binary, microseconds)
IMFLMNTM	long long int	Minimum elapsed time spent processing an input message (8 bytes binary, microseconds)
IMFLTPCP	long long int	Total processor time spent processing input messages (8 bytes binary, microseconds)
IMFLMXCP	long long int	Maximum processor time spent processing an input message (8 bytes binary, microseconds)
IMFLMNCP	long long int	Minimum processor time spent processing an input message (8 bytes binary, microseconds)
IMFLWTCP	long long int	Total processor time spent waiting for input messages (8 bytes binary, microseconds)
IMFLWTIN	long long int	Total elapsed time spent waiting for input messages (8 bytes binary, microseconds)
IMFLTPMG	unsigned int	Total number of messages processed
IMFLTSMG	long long int	Total size of input messages (bytes)
IMFLMXMG	long long int	Maximum input message size (bytes)
IMFLMNMG	long long int	Minimum input message size (bytes)
IMFLTHDP	unsigned int	Number of threads in pool
IMFLTHDM	unsigned int	Number of times the maximum number of threads is reached
IMFLERMQ[1]	unsigned int	Number of MQGET errors (MQInput node) or Web services errors (HTTPInput node)
IMFLERMG[2]	unsigned int	Number of messages that contain errors
IMFLERPR	unsigned int	Number of errors processing a message
IMFLTMOU	unsigned int	Number of timeouts processing a message (AggregateReply node only)
IMFLCMIT	unsigned int	Number of transaction commits
IMFLBKOU	unsigned int	Number of transaction backouts
IMFLACCT	char[32]	Accounting origin

Notes:
1. For example, a conversion error occurs when the message is got from the queue.
2. These include exceptions that are thrown downstream of the input node, and errors detected by the input node after it has successfully retrieved the message from the queue (for example, a format error).

The following table describes the contents of the BipSMFThread record.

Field	Data type	Details
ITHDID	short int	Control block hex ID (BipSMFThread_ID)
ITHDLEN	short int	Length of control block
ITHDEYE	char[4]	Eyecatcher (ITHD)
ITHDVER	int	Version number (BipSMFRecordVersion)
ITHDNBR	unsigned int	Relative thread number in pool

Field	Data type	Details
ITHDTPMG	unsigned int	Total number of messages processed by thread
ITHDTPTM	long long int	Total elapsed time spent processing input messages (8 bytes binary, microseconds)
ITHDTPCP	long long int	Total processor time spent processing input messages (8 bytes binary, microseconds)
ITHDWTCP	long long int	Total processor time spent waiting for input messages (8 bytes binary, microseconds)
ITHDWTIN	long long int	Total elapsed time spent waiting for input messages (8 bytes binary, microseconds)
ITHDTSMG	long long int	Total size of input messages (bytes)
ITHDMXMG	long long int	Maximum size of input messages (bytes)
ITHDMNMG	long long int	Minimum size of input messages (bytes)

The following table describes the contents of the BipSMFNode record.

Field	Data type	Details
INODID	short int	Control block hex ID (BipSMFNode_ID)
INODLEN	short int	Length of control block
INODEYE	char[4]	Eyecatcher (INOD)
INODVER	int	Version number (BipSMFRecordVersion)
INODNDNM	char[32]	Name of node (Label)
INODTYPE	char[32]	Type of node
INODTPTM	long long int	Total elapsed time spent processing input messages (8 bytes binary, microseconds)
INODMXTM	long long int	Maximum elapsed time spent processing input messages (8 bytes binary, microseconds)
INODMNTM	long long int	Minimum elapsed time spent processing input messages (8 bytes binary, microseconds)
INODTPCP	long long int	Total processor time spent processing input messages (8 bytes binary, microseconds)
INODMXCP	long long int	Maximum processor time spent processing input messages (8 bytes binary, microseconds)
INODMNCP	long long int	Minimum processor time spent processing input messages (8 bytes binary, microseconds)
INODTPMG	unsigned int	Total number of messages processed by this node
INODNITL	unsigned int	Number of input terminals
INODNOTL	unsigned int	Number of output terminals

The following table describes the contents of the BipSMFTerminal record.

Field	Data type	Details
ITRMID	short int	Control block hex ID (BipSMFTerminal_ID)
ITRMLEN	short int	Length of control block
ITRMEYE	char[4]	Eyecatcher (ITRM)
ITRMVER	int	Version number (BipSMFRecordVersion)

Field	Data type	Details
ITRMTLNM	char[32]	Name of terminal
ITRMTYPE	char[8]	Type of terminal, one of: • Input • Output
ITRMTINV	unsigned int	Total number of invocations

Example message flow accounting and statistics data

You can view message flow accounting and statistic data in an XML publication or user trace entries. To view z/OS SMF records, use a utility program that processes SMF records.

The following topics give example output in two formats:
- XML publication
- User trace entries

An example is not provided for z/OS SMF records, because these contain hexadecimal data and are not easily viewed in that form. To view SMF records, use any available utility program that processes SMF records.

Example of an XML publication for message flow accounting and statistics

This example shows an XML publication that contains message flow accounting and statistics data.

The following example shows what is generated for a snapshot report. The content of this publication message shows that the message flow is called *XMLflow*, and that it is running in an execution group named *default* on broker *MQ02BRK*. The message flow contains the following nodes:
- An MQInput node called *INQueue3*
- An MQOutput node called *OUTQueue*
- An MQOutput node called *FAILQueue*

The MQInput node's Out terminal is connected to the OUTQueue node. The MQInput node's Failure terminal is connected to the FAILQueue node.

During the interval for which statistics have been collected, this message flow processed no messages.

A publication that is generated for this data always includes the appropriate folders, even if there is no current data.

The following command has been issued to achieve these results:

```
mqsichangeflowstats MQ02BRK -s -c active -e default -f XMLFlow -n advanced -t basic -b basic -o xml
```

Blank lines have been added between folders to improve readability.

The broker takes information about statistics and accounting from the operating system. On some operating systems, such as Windows, UNIX, and Linux, rounding can occur because the system calls that are used to determine the processor times are not sufficiently granular. This rounding might affect the accuracy of the data.

The following example is the subscription message. The <psc> and <mcd> elements are part of the RFH header.

```
<psc>
  <Command>Publish</Command>
  <PubOpt>RetainPub</PubOpt>
  <Topic>$SYS/Broker/MQ02BRK/StatisticsAccounting/SnapShot/default/XMLflow
  </Topic>
</psc>


<mcd>
  <Msd>xml</Msd>
</mcd>
```

The following example is the publication that the broker generates:

```
<WMQIStatisticsAccounting RecordType="SnapShot" RecordCode="Snapshot">

<MessageFlow BrokerLabel="MQ02BRK"
 BrokerUUID="7d951e31-f200-0000-0080-efe1b9d849dc"
 ExecutionGroupName="default"
 ExecutionGroupUUID="77cf1e31-f200-0000-0080-efe1b9d849dc"
 MessageFlowName="XMLflow" StartDate="2003-01-17"
 StartTime="14:44:34.581320" EndDate="2003-01-17" EndTime="14:44:44.582926"
 TotalElapsedTime="0"
 MaximumElapsedTime="0" MinimumElapsedTime="0" TotalCPUTime="0"
 MaximumCPUTime="0"MinimumCPUTime="0" CPUTimeWaitingForInputMessage="685"
 ElapsedTimeWaitingForInputMessage="10001425" TotalInputMessages="0"
 TotalSizeOfInputMessages="0" MaximumSizeOfInputMessages="0"
 MinimumSizeOfInputMessages="0" NumberOfThreadsInPool="1"
 TimesMaximumNumberOfThreadsReached="0" TotalNumberOfMQErrors="0"
 TotalNumberOfMessagesWithErrors="0" TotalNumberOfErrorsProcessingMessages="0"
 TotalNumberOfTimeOutsWaitingForRepliesToAggregateMessages="0"
 TotalNumberOfCommits="0" TotalNumberOfBackouts="0" AccoutingOrigin="DEPT1"/>


<Threads Number="1">
<ThreadStatistics Number="5" TotalNumberOfInputMessages="0"
TotalElapsedTime="0" TotalCPUTime="0" CPUTimeWaitingForInputMessage="685"
ElapsedTimeWaitingForInputMessage="10001425" TotalSizeOfInputMessages="0"
MaximumSizeOfInputMessages="0" MinimumSizeOfInputMessages="0"/>
</Threads>


<Nodes Number="3">


  <NodeStatistics Label="FAILQueue" Type="MQOutput" TotalElapsedTime="0"
    MaximumElapsedTime="0" MinimumElapsedTime="0" TotalCPUTime="0"
    MaximumCPUTime="0" MinimumCPUTime="0" CountOfInvocations="0"
    NumberOfInputTerminals="1" NumberOfOutputTerminals="2">
   <TerminalStatistics Label="failure" Type="Output" CountOfInvocations="0"/>
   <TerminalStatistics Label="in" Type="Input" CountOfInvocations="0"/>
   <TerminalStatistics Label="out" Type="Output" CountOfInvocations="0"/>
  </NodeStatistics>


  <NodeStatistics Label="INQueue3" Type="MQInput" TotalElapsedTime="0"
    MaximumElapsedTime="0" MinimumElapsedTime="0" TotalCPUTime="0"
    MaximumCPUTime="0" MinimumCPUTime="0"CountOfInvocations="0"
    NumberOfInputTerminals="0" NumberOfOutputTerminals="3">
   <TerminalStatistics Label="catch" Type="Output" CountOfInvocations="0"/>
   <TerminalStatistics Label="failure" Type="Output" CountOfInvocations="0"/>
   <TerminalStatistics Label="out" Type="Output" CountOfInvocations="0"/>
  </NodeStatistics>
```

```
  <NodeStatistics Label="OUTQueue" Type="MQOutput" TotalElapsedTime="0"
    MaximumElapsedTime="0" MinimumElapsedTime="0" TotalCPUTime="0"
    MaximumCPUTime="0" MinimumCPUTime="0" CountOfInvocations="0"
    NumberOfInputTerminals="1" NumberOfOutputTerminals="2">
    <TerminalStatistics Label="failure" Type="Output" CountOfInvocations="0"/>
    <TerminalStatistics Label="in" Type="Input" CountOfInvocations="0"/>
    <TerminalStatistics Label="out" Type="Output" CountOfInvocations="0"/>
   </NodeStatistics>


 </Nodes>


</WMQIStatisticsAccounting>
```

Example of user trace entries for message flow accounting and statistics

This example shows a user trace that contains message flow accounting and statistics data.

The following example shows what is generated for a snapshot report. The messages that are written to the trace show that the message flow is called *myExampleFlow*, and that it is running in an execution group named *default* on broker *MQ01BRK*. The message flow contains the following nodes:

- An MQInput node called *inNode*
- A Compute node called *First1*
- An MQOutput node called *outNode*

The nodes are connected together (Out terminal to In terminal for each connection).

During the interval for which statistics have been collected, this message flow processed 150 input messages.

The records show that two threads are assigned to this message flow. One thread is assigned when the message flow is deployed (the default number); an additional thread (thread 0) listens on the input queue. The listening thread starts additional threads to process input messages that are dependent on the number of instances that you have configured for the message flow, and on the rate of arrival of the input messages on the input queue.

The following command has been issued to achieve these results:

```
mqsichangeflowstats MQ01BRK -s -c active -e default -f myExampleFlow -n advanced -t basic -b basic
```

The trace entries have been retrieved with the mqsireadlog command and formatted using the mqsiformatlog command. The output from mqsiformatlog is shown in the following example. Line breaks have been added to aid readability.

The broker takes information about statistics and accounting from the operating system. On some operating systems, such as Windows, UNIX, and Linux, rounding can occur because the system calls that are used to determine the processor times are not sufficiently granular. This rounding might affect the accuracy of the data.

```
BIP2380I: WMQI message flow statistics. ProcessID='328467', Key='6', Type='SnapShot', Reason='Snapshot',
BrokerLabel='MQ01BRK', BrokerUUID='18792e66-e100-0000-0080-f197e5ed81bd',
ExecutionGroupName='default', ExecutionGroupUUID='15d4314a-3607-11d4-8000-09140f7b0000',
MessageFlowName='myExampleFlow',
StartDate='2003-05-20', StartTime='13:44:31.885862',
EndDate='2003-05-20', EndTime='13:44:51.310080',
TotalElapsedTime='9414843', MaximumElapsedTime='1143442', MinimumElapsedTime='35154',
```

```
TotalCPUTime='760147', MaximumCPUTime='70729', MinimumCPUTime='3124',
CPUTimeWaitingForInputMessage='45501', ElapsedTimeWaitingForInputMessage='11106438',
TotalInputMessages='150', TotalSizeOfInputMessages='437250',
MaximumSizeOfInputMessages='2915', MinimumSizeOfInputMessages='2915',
NumberOfThreadsInPool='1', TimesMaximumNumberOfThreadsReached='150',
TotalNumberOfMQErrors='0', TotalNumberOfMessagesWithErrors='0',
TotalNumberOfErrorsProcessingMessages='0', TotalNumberOfTimeOuts='0',
TotalNumberOfCommits='150', TotalNumberOfBackouts='0', AccountingOrigin="DEPT2".
Statistical information for message flow 'myExampleFlow' in broker 'MQ01BRK'.
This is an information message produced by WMQI statistics.

BIP2381I: WMQI thread statistics. ProcessID='328467', Key='6', Number='0',
TotalNumberOfInputMessages='0',
TotalElapsedTime='0', TotalCPUTime='0', CPUTimeWaitingForInputMessage='110',
ElapsedTimeWaitingForInputMessage='5000529', TotalSizeOfInputMessages='0',
MaximumSizeOfInputMessages='0', MinimumSizeOfInputMessages='0'.
Statistical information for thread '0'.
This is an information message produced by WMQI statistics.

BIP2381I: WMQI thread statistics. ProcessID='328467', Key='6', Number='18',
TotalNumberOfInputMessages='150',
TotalElapsedTime='9414843', TotalCPUTime='760147', CPUTimeWaitingForInputMessage='45391',
ElapsedTimeWaitingForInputMessage='6105909', TotalSizeOfInputMessages='437250',
MaximumSizeOfInputMessages='2915', MinimumSizeOfInputMessages='2915'.
Statistical information for thread '18'.
This is an information message produced by WMQI statistics.

BIP2382I: WMQI node statistics. ProcessID='328467', Key='6',
Label='First1', Type='ComputeNode',
TotalElapsedTime='6428815', MaximumElapsedTime='138261', MinimumElapsedTime='28367',
TotalCPUTime='604060', MaximumCPUTime='69645', MinimumCPUTime='2115',
CountOfInvocations='150', NumberOfInputTerminals='1', NumberOfOutputTerminals='2'.
Statistical information for node 'First1'.
This is an information message produced by WMQI statistics.

BIP2383I: WMQI terminal statistics. ProcessID='328467', Key='6',
Label='failure', Type='Output', CountOfInvocations='0',
Statistical information for terminal 'failure'.
This is an information message produced by WMQI statistics.

BIP2383I: WMQI terminal statistics. ProcessID='328467', Key='6',
Label='in', Type='Input', CountOfInvocations='150',
Statistical information for terminal 'in'.
This is an information message produced by WMQI statistics.

BIP2383I: WMQI terminal statistics. ProcessID='328467', Key='6',
Label='out', Type='Output', CountOfInvocations='150',
Statistical information for terminal 'out'.
This is an information message produced by WMQI statistics.

BIP2382I: WMQI node statistics. ProcessID='328467', Key='6',
Label='inNode', Type='MQInputNode',
TotalElapsedTime='1813446', MaximumElapsedTime='1040209', MinimumElapsedTime='1767',
TotalCPUTime='70565', MaximumCPUTime='686', MinimumCPUTime='451',
CountOfInvocations='150', NumberOfInputTerminals='0', NumberOfOutputTerminals='3'.
Statistical information for node 'inNode'.
This is an information message produced by WMQI statistics.

BIP2383I: WMQI terminal statistics. ProcessID='328467', Key='6',
Label='catch', Type='Output', CountOfInvocations='0',
Statistical information for terminal 'catch'.
This is an information message produced by WMQI statistics.

BIP2383I: WMQI terminal statistics. ProcessID='328467', Key='6',
Label='failure', Type='Output', CountOfInvocations='0',
Statistical information for terminal 'failure'.
This is an information message produced by WMQI statistics.
```

```
BIP2383I: WMQI terminal statistics. ProcessID='328467', Key='6',
Label='out', Type='Output', CountOfInvocations='150',
Statistical information for terminal 'out'.
This is an information message produced by WMQI statistics.

BIP2382I: WMQI node statistics. ProcessID='328467', Key='6',
Label='outNode', Type='MQOutputNode',
TotalElapsedTime='1172582', MaximumElapsedTime='177516', MinimumElapsedTime='3339',
TotalCPUTime='85522', MaximumCPUTime='762', MinimumCPUTime='536',
CountOfInvocations='150', NumberOfInputTerminals='1', NumberOfOutputTerminals='2'.
Statistical information for node 'outNode'.
This is an information message produced by WMQI statistics.

BIP2383I: WMQI terminal statistics. ProcessID='328467', Key='6',
Label='failure', Type='Output', CountOfInvocations='0',
Statistical information for terminal 'failure'.
This is an information message produced by WMQI statistics.

BIP2383I: WMQI terminal statistics. ProcessID='328467', Key='6',
Label='in', Type='Input', CountOfInvocations='150',
Statistical information for terminal 'in'.
This is an information message produced by WMQI statistics.

BIP2383I: WMQI terminal statistics. ProcessID='328467', Key='6',
Label='out', Type='Output', CountOfInvocations='0',
Statistical information for terminal 'out'.
This is an information message produced by WMQI statistics.
```

Coordinated message flows

A coordinated message flow runs in a single transaction, which is started when a message is received by an input node, and can be committed or rolled back when all processing has completed.

The following topics provide reference information for database use in coordinated message flows:

- "Database connections for coordinated message flows"
- "Database support for coordinated message flows" on page 1517

Database connections for coordinated message flows

When you configure a message flow to access a database, the broker establishes a connection to that database based on the ODBC DSN.

When you configure a message flow to access a database, the broker establishes a connection to that database based on the ODBC or JDBC DSN. To coordinate the database updates with other updates (determined by the configuration you have set for each node that accesses a database), the broker makes a connection for each transaction mode for each DSN accessed on each message flow thread.

Therefore, if you set the Transaction Mode property for one node in the message flow to Automatic, and for another node to Commit, the broker establishes two separate connections to this DSN from the same thread. Take this action into account when you calculate the number of connections required between a broker and a specific DSN.

For further information about connections made by the broker to user databases, see User database connections.

Connections to user databases are in addition to the runtime connections that are required by the broker (to the DB2, Informix, Oracle, Sybase, or SQL Server

database that is defined to hold its internal information). For details of these connections, refer to Enabling ODBC connections to the databases and Enabling JDBC connections to the databases.

Database support for coordinated message flows

If the message flow processing includes interaction with an external database, you can coordinate the transaction by using XA technology.

XA coordination ensures that all participants update or return to a consistent state. This external coordination support is provided by the underlying WebSphere MQ facilities on distributed systems, and by Resource Recovery Services (RRS) on z/OS.

The following databases provide the correct level of XA support for coordinating message flows on distributed systems:

- DB2
- Oracle
- Sybase

On z/OS, database support for coordinated message flows is provided by DB2 only.

Element definitions for message parsers

The topics in this section discuss data types for the WebSphere MQ headers, and define the element names, types, and attributes for each of the supported headers:

For each parser, the following terms are defined:

- Root element name: the name of the syntax element created by the parser at the root of its own part of the tree.
- Class name: the name by which the parser defines itself to WebSphere Message Broker.

Data types of fields and elements

The fields within WebSphere MQ headers and other subtrees built from the message are of a particular data type. When you manipulate the messages and their headers using ESQL in the message flow nodes, be aware of type information in field references:

- "Data types for elements in the Properties subtree"
- "Data types for elements in the MQ DestinationData subtree" on page 1519
- Data types for elements in an MRM message
- "Data types for an unstructured (BLOB) message" on page 1522
- Field names of the IDOC parser structures

Data types of the fields in the WebSphere MQ headers

The fields in the WebSphere MQ headers have specific data types. Parsers are supplied for the headers included in WebSphere MQ messages.

The parsers determine the data type of each field in the header:
- "The MQCFH parser" on page 1522
- "The MQCIH parser" on page 1522
- "The MQDLH parser" on page 1524
- "The MQIIH parser" on page 1524
- "The MQMD parser" on page 1525
- "The MQMDE parser" on page 1526
- "The MQRFH parser" on page 1527
- "The MQRFH2 and MQRFH2C parsers" on page 1527
- "The MQRMH parser" on page 1527
- "The MQSAPH parser" on page 1528
- "The MQWIH parser" on page 1529
- "The SMQ_BMH parser" on page 1529

The mapping of the WebSphere MQ data types to the data types used in the broker is shown in the following table.

Data type of the field	Represented as
MQLONG	INTEGER
MQCHAR, MQCHAR4	CHARACTER
MQBYTE, MQBYTEn	BLOB

Data types for elements in the Properties subtree

A parser is supplied for the Properties subtree; it associates each field with a specific data type.

The fields and data type of each field are shown in the following table.

Data type of the element	Represented as
CodedCharSetId	INTEGER
CreationTime	TIMESTAMP
ContentType	CHARACTER
Encoding	INTEGER
ExpirationTime	TIMESTAMP
IdentityMappedIssuedBy	CHARACTER
IdentityMappedPassword	CHARACTER
IdentityMappedToken	CHARACTER
IdentityMappedType	CHARACTER
IdentitySourceIssuedBy	CHARACTER
IdentitySourcePassword	CHARACTER

Data type of the element	Represented as
IdentitySourceToken	CHARACTER
IdentitySourceType	CHARACTER
MessageFormat	CHARACTER
MessageSet	CHARACTER
MessageType	CHARACTER
Persistence	BOOLEAN
Priority	INTEGER
ReplyIdentifier	CHARACTER
ReplyProtocol	CHARACTER
Topic (this field contains a list)	CHARACTER
Transactional	BOOLEAN

Data types for elements in the MQ DestinationData subtree

The DestinationData subtree is part of the Destination subtree in the local environment. Local environment trees are created by input nodes when they receive a message and, optionally, by Compute nodes. When created, the trees are empty but you can create data in them by using ESQL statements coded in any of the SQL nodes.

The Destination subtree consists of subtrees for zero or more protocols; for example, WebSphere MQ and WebSphere MQ Everyplace, or a subtree for routing destinations (RouterList), or both.

The protocol tree has two children:

- Defaults is the first element; there can be only one.
- DestinationData is the following element, and can be repeated any number of times, to represent each destination to which a message is sent.

"Local environment tree structure" on page 80 includes a picture of a typical tree, showing a Destination tree that has both protocol and RouterList subtrees.

The structure of data within the DestinationData folder is the same as that in Defaults for the same protocol, and can be used to override the default values in Defaults. You can therefore set up Defaults to contain values that are common to all destinations, and set only the unique values in each DestinationData subtree. If a value is set neither in DestinationData, nor in Defaults, the value that you have set for the corresponding node property is used.

The fields, data type, and valid values for each element of Defaults and DestinationData subtrees for WebSphere MQ are shown in the following table. "MQOutput node" on page 1100 describes the corresponding node properties.

Refer to "Accessing the local environment tree" on page 353 for information about using DestinationData.

Data type of the element	Represented as	Corresponding node property	Valid values
queueManagerName	CHARACTER	Queue Manager Name	
queueName	CHARACTER	Queue Name	

Data type of the element	Represented as	Corresponding node property	Valid values
transactionMode	CHARACTER	Transaction Mode	no, yes, automatic
persistenceMode	CHARACTER	Persistence Mode	no, yes, automatic, asQdef
newMsgId	CHARACTER	New Message ID	no, yes
newCorrelId	CHARACTER	New Correlation ID	no, yes
segmentationAllowed	CHARACTER	Segmentation Allowed	no, yes
alternateUserAuthority	CHARACTER	Alternate User Authority	no, yes
replyToQMgr	CHARACTER	Reply-to queue manager	
replyToQ	CHARACTER	Reply-to queue	

Case-sensitivity for data types and values

When you create these fields in the DestinationData folder, enter the data type and value exactly as shown in the table. If any variations in spelling or case are used, these fields or values are ignored in the DestinationData records and the next available value is used.

For example, the following ESQL samples could result in unexpected output:

```
SET OutputLocalEnvironment.Destination.MQ.DestinationData[1].persistenceMode = 'YES';
SET OutputLocalEnvironment.Destination.MQ.DestinationData[2].PersistenceMode = 'yes';
```

In each case, the DestinationData folder might not write a persistent message for these destinations. In the first example, the persistenceMode field has been given a value of 'YES', which is not one of the valid values listed in the table, and this value is ignored. In the second example, the field named 'PersistenceMode' is specified incorrectly and is ignored. Either the persistenceMode value of the Defaults folder, or the value of the associated attribute on the MQOutput node are used. If this behavior causes a value of 'no' or 'automatic' to be used, a persistent message is not written.

If a DestinationData folder is producing unexpected output, check that you have used the correct case and spelling in the fields and values.

Using LocalEnvironment variables with JMSOutput and JMSReply nodes

The LocalEnvironment data elements related to the processing of JMS Messages in the JMSOutput and JMSReply nodes.

LocalEnvironment.WrittenDestination.JMS.DestinationData fields

The DestinationData element is a data structure created by the JMSOutput and JMSReply nodes for each message that is sent to a JMS Queue or published to a JMS Topic when:

- The Out terminal of the node is connected to another node in the message flow
- An output message callback routine has been enabled for the message flow. See cciOutputMessageCallback.

The fields in the DestinationData structure are described in the following table, and can be used by a receiving application, or an output message callback routine, to link request messages with reply messages:

Element name	Element Data Type	Description
destinationName	CHARACTER	The name of the JMS Queue to which the node sends an outgoing message or the JMS Topic to which the node publishes. This value can specified in these formats: • a JNDI administered object (predefined in the JNDI bindings specified in the Location JNDI bindings node property). In which case the format of the value is `jndi://<JNDI Object name>` • a character string that represents the JMS Destination name in an internal format that recognized by that particular JMS provider. For example, when using WebSphere MQ as the JMS provider a JMS Queue Destination would be represented by the character string `queue://<queue manager>/<queue name>`
initialContext	CHARACTER	The Java Class name of the Initial Context Factory for the JMS provider that the JMSOutput or JMSReply node connects to.
JMSMessageID	CHARACTER	A JMS message ID is the value assigned by a JMS Provider when a message is sent to a JMS Queue or published to a JMS Topic. This value is retrieved from the JMS Message object after the message is sent or published.
JMSCorrelationID	CHARACTER	The JMS Message header property called JMSCorrelationID can be used to hold a value referencing some external information to be used for linking request with reply messages. When creating the DestinationData element in the LocalEnvironment this correlation ID value is obtained from the JMSCorrelationID message tree field in the folder OutputRoot.JMSTransport.Transport_Folders. Header_Values.

LocalEnvironment.Destination.JMSDestinationList fields

Transformation nodes can write data elements called DestinationData[n] in the LocalEnvironment folder called Destination.JMSDestinationList. The DestinationData elements are written with an array subscript format where the subscript is an integer that identifies an individual element in the destination list.

A JMSOutput node searches for DestinationData[n] entries in the LocalEnvironment if it has been configured to send to a destination list. The node sends an output message to each entry found in the destination list. The following table describes the format of a DestinationData element.

Element name	Element Data Type	Description
DestinationData	CHARACTER	The name of the JMS Queue to which the node sends an outgoing message or the JMS Topic to which the node publishes. This value can specified in these formats: • a JNDI administered object (predefined in the JNDI bindings specified in the Location JNDI bindings node property). In which case the format of the value is `jndi://<JNDI Object name>` • a character string that represents the JMS Destination name in an internal format that recognized by that particular JMS provider. For example, when using WebSphere MQ as the JMS provider a JMS Queue Destination would be represented by the character string `queue://<queue manager>/<queue name>`

Data types for an unstructured (BLOB) message

A parser is supplied for the body of a message in the BLOB domain; it associates each field with a specific data type.

An unstructured (BLOB) message has the data types shown in the following table.

Data type of the element	Represented as
BLOB	BLOB
UnknownParserName	CHARACTER

If the broker cannot find a parser that corresponds to the domain that is requested by the user, the message is assigned to the BLOB parser, and the requested domain is preserved in the *UnknownParserName* field. If a BLOB parser is explicitly created, the UnknownParserName field is still present. It can contain the values of "BLOB" or "none" or can be the zero length string ("").

This information is used by the header integrity routine (described in “Parsers” on page 91) to ensure that the semantic meaning of the message is preserved.

The MQCFH parser

The elements of the MQCFH parser are listed in this topic.

The root name for this parser is MQPCF. The class name is MQPCF.

The following table lists the elements native to the MQCFH header.

Element Name	Element Data Type	Element Attributes
Type	INTEGER	Name Value
StrucLength	INTEGER	Name Value
Version	INTEGER	Name Value
Command	INTEGER	Name Value
MsgSeqNumber	INTEGER	Name Value
Control	INTEGER	Name Value
CompCode	INTEGER	Name Value
Reason	INTEGER	Name Value
ParameterCount	INTEGER	Name Value

For further information about this header and its contents, see the *Programmable Command Formats and Administration Interface* section of the WebSphere MQ Version 6 Information Center online or WebSphere MQ Version 7 Information Center online.

The MQCIH parser

The elements of the MQCIH parser are listed in this topic.

The root name for this parser is MQCIH. The class name is MQCICS.

The following table lists the elements native to the MQCIH header.

Element Name	Element Data Type	Element Attributes
Format	CHARACTER	Name Value
Version	INTEGER	Name Value
Encoding	INTEGER	Name Value
CodedCharSetId	INTEGER	Name Value
Flags	INTEGER	Name Value
ReturnCode	INTEGER	Name Value
CompCode	INTEGER	Name Value
Reason	INTEGER	Name Value
UOWControl	INTEGER	Name Value
GetWaitInterval	INTEGER	Name Value
LinkType	INTEGER	Name Value
OutputDataLength	INTEGER	Name Value
FacilityKeepTime	INTEGER	Name Value
ADSDescriptor	INTEGER	Name Value
ConversationalTask	INTEGER	Name Value
TaskEndStatus	INTEGER	Name Value
Facility	BLOB	Name Value
Function	CHARACTER	Name Value
AbendCode	CHARACTER	Name Value
Authenticator	CHARACTER	Name Value
Reserved1	CHARACTER	Name Value
ReplyToFormat	CHARACTER	Name Value
RemoteSysId	CHARACTER	Name Value
RemoteTransId	CHARACTER	Name Value
TransactionId	CHARACTER	Name Value
FacilityLike	CHARACTER	Name Value
AttentionId	CHARACTER	Name Value
StartCode	CHARACTER	Name Value
CancelCode	CHARACTER	Name Value
NextTransactionId	CHARACTER	Name Value
Reserved2	CHARACTER	Name Value
Reserved3	CHARACTER	Name Value
CursorPosition	INTEGER	Name Value
ErrorOffset	INTEGER	Name Value
InputItem	INTEGER	Name Value
Reserved4	INTEGER	Name Value

For further information about this header and its contents, see the *Application Programming Reference* section of the WebSphere MQ Version 6 Information Center online or WebSphere MQ Version 7 Information Center online.

The MQDLH parser

The elements of the MQDLH parser are listed in this topic.

The root name for this parser is MQDLH. The class name is MQDEAD.

The following table lists the elements native to the MQDLH header.

Element Name	Element Data Type	Element Attributes
Format	CHARACTER	Name Value
Version	INTEGER	Name Value
Encoding	INTEGER	Name Value
CodedCharSetId	INTEGER	Name Value
Reason	INTEGER	Name Value
DestQName	CHARACTER	Name Value
DestQMgrName	CHARACTER	Name Value
PutApplType	INTEGER	Name Value
PutApplName	CHARACTER	Name Value
PutDate	TIMESTAMP/CHARACTER	Name Value
PutTime	TIMESTAMP/CHARACTER	Name Value

For further information about this header and its contents, see the *Application Programming Reference* section of the WebSphere MQ Version 6 Information Center online or WebSphere MQ Version 7 Information Center online.

The MQIIH parser

The elements of the MQIIH parser are listed in this topic.

The root name for this parser is MQIIH. The class name is MQIMS.

The following table lists the elements native to the MQIIH header.

Element Name	Element Data Type	Element Attributes
Format	CHARACTER	Name Value
Version	INTEGER	Name Value
Encoding	INTEGER	Name Value
CodedCharSetId	INTEGER	Name Value
Flags	INTEGER	Name Value
LTermOverride	CHARACTER	Name Value
MFSMapName	CHARACTER	Name Value
ReplyToFormat	CHARACTER	Name Value
Authenticator	CHARACTER	Name Value
TranInstanceId	BLOB	Name Value
TranState	CHARACTER	Name Value
CommitMode	CHARACTER	Name Value
SecurityScope	CHARACTER	Name Value
Reserved	CHARACTER	Name Value

For further information about this header and its contents, see the *Application Programming Reference* section of the WebSphere MQ Version 6 Information Center online or WebSphere MQ Version 7 Information Center online.

The MQMD parser

The elements of the MQMD parser are listed in this topic.

The root name for this parser is MQMD. The class name is MQHMD.

The following table lists the orphan elements adopted by the MQMD header.

Element Name	Element Data Type	Element Attributes
SourceQueue	CHARACTER	Name Value
Transactional	BOOLEAN	Name Value

The following table lists the elements native to the MQMD header.

Element Name	Element Data Type	Element Attributes
Format	CHARACTER	Name Value
Version	INTEGER	Name Value
Report	INTEGER	Name Value
MsgType	INTEGER	Name Value
Expiry[1]	INTEGER/GMTTIMESTAMP	Name Value
Feedback	INTEGER	Name Value
Encoding	INTEGER	Name Value
CodedCharSetId	INTEGER	Name Value
Priority	INTEGER	Name Value
Persistence	INTEGER	Name Value
MsgId	BLOB	Name Value
CorrelId	BLOB	Name Value
BackoutCount	INTEGER	Name Value
ReplyToQ	CHARACTER	Name Value
ReplyToQMgr	CHARACTER	Name Value
UserIdentifier	CHARACTER	Name Value
AccountingToken	BLOB	Name Value
ApplIdentityData	CHARACTER	Name Value
PutApplType	INTEGER	Name Value
PutApplName	CHARACTER	Name Value
PutDate	TIMESTAMP/CHARACTER	Name Value
PutTime	TIMESTAMP/CHARACTER	Name Value
ApplOriginData	CHARACTER	Name Value
GroupId	BLOB	Name Value
MsgSeqNumber	INTEGER	Name Value

Element Name	Element Data Type	Element Attributes
Offset	INTEGER	Name Value
MsgFlags	INTEGER	Name Value
OriginalLength	INTEGER	Name Value

Note:

1. The Expiry field in the MQMD is a special case:
   - An INTEGER value represents an expiry interval in tenths of a second. If the Expiry field is set to -1, it represents an unlimited expiry interval (that is, the message never expires) If the Expiry field is a positive INTEGER, it represents an expiry interval of that number of tenths of a second (for example, if it is set to 4, it represents 4 tenths of a second, if it is set to 15, it represents one and a half seconds).
   - A GMTTIMESTAMP value represents a specific expiration time.

   If Expiry contains a GMTTIMESTAMP in the past, or an INTEGER of less than 1 (excluding -1), it is set to the value 1 (one tenth of a second, the minimum value).

For further information about this header and its contents, see the *Application Programming Reference* section of the WebSphere MQ Version 6 Information Center online or WebSphere MQ Version 7 Information Center online.

The MQMDE parser

The elements of the MQMDE parser are listed in this topic.

The root name for this parser is MQMDE. The class name is MQHMDE.

The following table lists the elements native to the MQMDE header.

Element Name	Element Data Type	Element Attributes
Format	CHARACTER	Name Value
Version	INTEGER	Name Value
Encoding	INTEGER	Name Value
CodedCharSetId	INTEGER	Name Value
Flags	INTEGER	Name Value
GroupId	BLOB	Name Value
MsgSeqNumber	INTEGER	Name Value
Offset	INTEGER	Name Value
MsgFlags	INTEGER	Name Value
OriginalLength	INTEGER	Name Value

For further information about this header and its contents, see the *Application Programming Reference* section of the WebSphere MQ Version 6 Information Center online or WebSphere MQ Version 7 Information Center online.

The MQRFH parser

The elements of the MQRFH parser are listed in this topic.

The root name for this parser is MQRFH. The class name is MQHRF.

The following table lists the elements native to the MQRFH header.

Element Name	Element Data Type	Element Attributes
Format	CHARACTER	Name Value
Version	INTEGER	Name Value
Encoding	INTEGER	Name Value
CodedCharSetId	INTEGER	Name Value
Flags	INTEGER	Name Value

Other name value elements might be present that contain information as parsed from or destined for the option buffer.

For further information about this header and its contents, see the *Application Programming Reference* section of the WebSphere MQ Version 6 Information Center online or WebSphere MQ Version 7 Information Center online.

The MQRFH2 and MQRFH2C parsers

The MQRFH2 header can be parsed using either the MQRFH2 parser or the MQRFH2C compact parser.

The root names for these parsers are MQRFH2 and MQRFH2C. The class names are MQHRF2 and MQHRF2C.

The following table lists the elements that are required for the MQRFH2 header.

Element Name	Element Data Type	Element Attributes
Format	CHARACTER	Name Value
Version	INTEGER	Name Value
Encoding	INTEGER	Name Value
CodedCharSetId	INTEGER	Name Value
Flags	INTEGER	Name Value
NameValueCCSID	INTEGER	Name Value

Other name and child name value elements might be present that contain information that is parsed from, or destined for, the option buffer. For further information about this header and its contents, see MQRFH2 header and the *Application Programming Reference* section of the WebSphere MQ Version 6 Information Center online or WebSphere MQ Version 7 Information Center online.

The MQRMH parser

The elements of the MQRMH parser are listed in this topic.

The root name for this parser is MQRMH. The class name is MQHREF.

The following table lists the elements native to the MQRMH header.

Element Name	Element Data Type	Element Attributes
Format	CHARACTER	Name Value
Version	INTEGER	Name Value
Encoding	INTEGER	Name Value
CodedCharSetId	INTEGER	Name Value
Flags	INTEGER	Name Value
ObjectType	CHARACTER	Name Value
ObjectInstanceId	BLOB	Name Value
SrcEnv	CHARACTER[1]	Name Value
SrcName	CHARACTER[2]	Name Value
DestEnv	CHARACTER[3]	Name Value
DestName	CHARACTER[4]	Name Value
DataLogicalLength	INTEGER	Name Value
DataLogicalOffset	INTEGER	Name Value
DataLogicalOffset2	INTEGER	Name Value

Notes:

1. This field represents both SrcEnvLength and Offset
2. This field represents both SrcNameLength and Offset
3. This field represents both DestEnvLength and Offset
4. This field represents both DestNameLength and Offset

For further information about this header and its contents, see the *Application Programming Reference* section of the WebSphere MQ Version 6 Information Center online or WebSphere MQ Version 7 Information Center online.

The MQSAPH parser

The elements of the MQSAPH parser are listed in this topic.

The root name for this parser is MQSAPH. The class name is MQHSAP.

The following table lists the elements native to the MQSAPH header.

Element Name	Element Data Type	Element Attributes
Format	CHARACTER	Name Value
Version	INTEGER	Name Value
Encoding	INTEGER	Name Value
CodedCharSetId	INTEGER	Name Value
Flags	INTEGER	Name Value
Client	CHARACTER	Name Value
Language	CHARACTER	Name Value
HostName	CHARACTER	Name Value
UserId	CHARACTER	Name Value
Password	CHARACTER	Name Value

Element Name	Element Data Type	Element Attributes
SystemNumber	CHARACTER	Name Value
Reserved	BLOB	Name Value

For further information about this header and its contents, see the WebSphere MQ Version 6 Information Center online or WebSphere MQ Version 7 Information Center online.

The MQWIH parser

The elements of the MQWIH parser are listed in this topic.

The root name for this parser is MQWIH. The class name is MQHWIH.

The following table lists the elements native to the MQWIH header.

Element Name	Element Data Type	Element Attributes
Format	CHARACTER	Name Value
Version	INTEGER	Name Value
Encoding	INTEGER	Name Value
CodedCharSetId	INTEGER	Name Value
Flags	INTEGER	Name Value
ServiceName	CHARACTER	Name Value
ServiceStep	CHARACTER	Name Value
MsgToken	BLOB	Name Value
Reserved	CHARACTER	Name Value

For further information about this header and its contents, see the *Application Programming Reference* section of the WebSphere MQ Version 6 Information Center online or WebSphere MQ Version 7 Information Center online.

The SMQ_BMH parser

The elements of the SMQ_BMH parser are listed in this topic.

The root name for this parser is SMQ_BMH. The class name is SMQBAD.

The following table lists the elements native to the SMQ_BMH header.

Element Name	Element Data Type	Element Attributes
Format	CHARACTER	Name Value
Version	INTEGER	Name Value
Encoding	INTEGER	Name Value
CodedCharSetId	INTEGER	Name Value
ErrorType	INTEGER	Name Value
Reason	INTEGER	Name Value
PutApplType	INTEGER	Name Value
PutApplName	CHARACTER	Name Value

Element Name	Element Data Type	Element Attributes
PutDate	TIMESTAMP/CHARACTER	Name Value
PutTime	TIMESTAMP/CHARACTER	Name Value

Message mappings

Edit and configure message maps using the Message Mapping editor.

This section contains topics that provide reference information about message mapping:

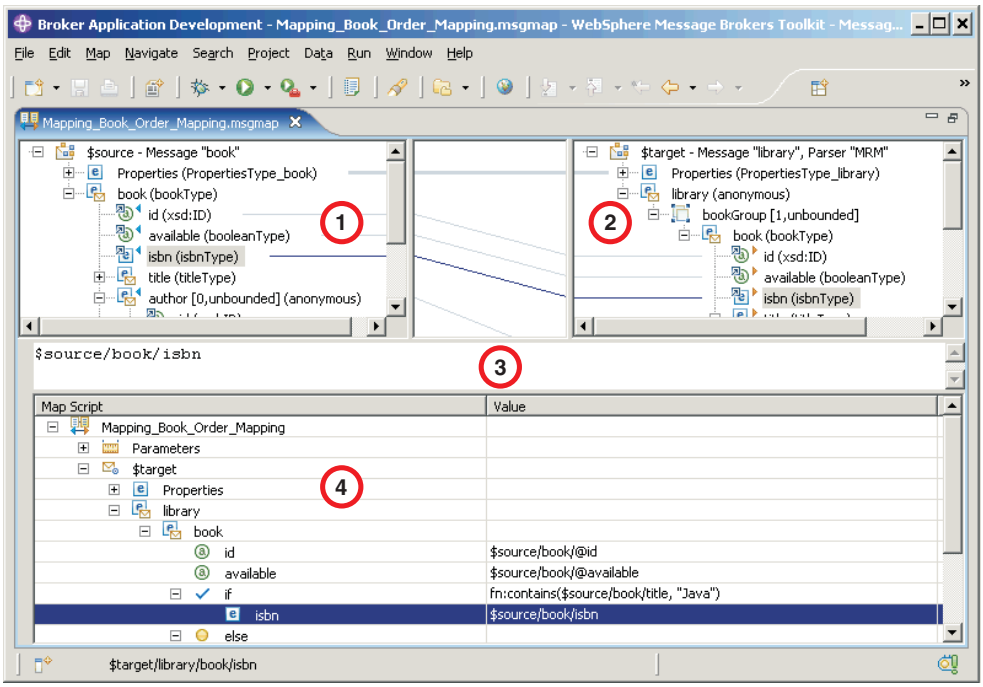
- "Message Mapping editor"
  - Source pane
  - Target pane
  - Edit pane
  - Spreadsheet pane
- "Mapping node" on page 1542
  - Syntax
  - Functions
  - Casts
- "Migrating message mappings from Version 5.0" on page 1562
  - Migration restrictions

Message Mapping editor

You configure a message mapping using the Message Mapping editor, which you use to set values for:

- the message destination
- message headers
- message content

Here is an example of the Message Mapping editor. There are separate panes for working with sources, targets and expressions, as well as a spreadsheet view.



1. **Source pane:** displays a source message or database table
2. **Target pane:** displays a target message
3. **Edit pane:** displays the expression to be used to derive the target element value
4. **Spreadsheet pane:** displays a summary of the mappings in spreadsheet columns (each target field and its value)

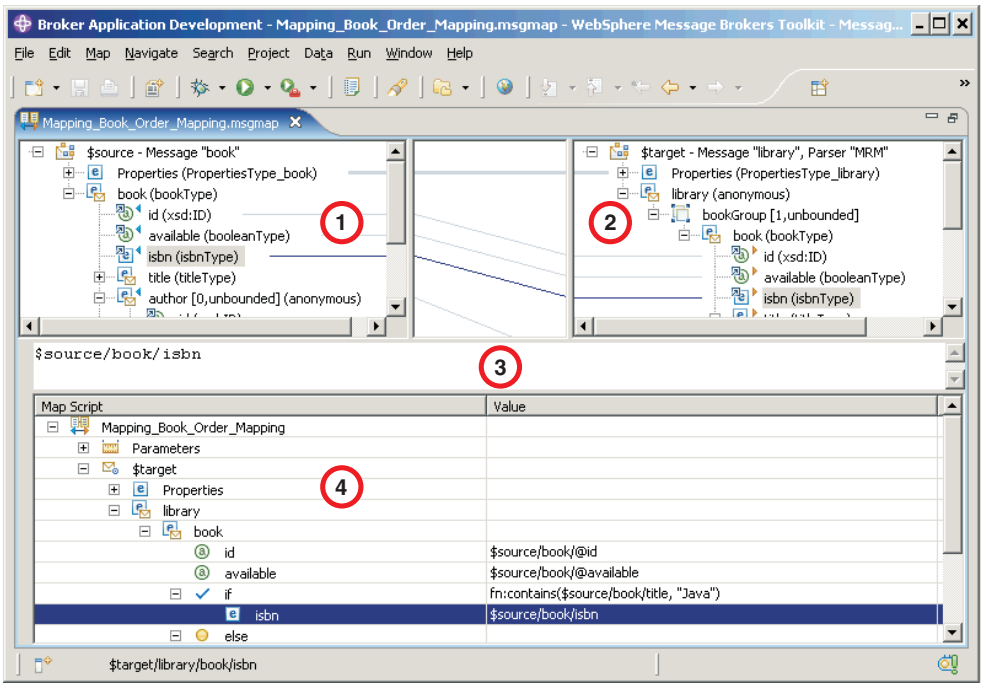
Use the Message Mapping editor to perform various mapping tasks.

Wizards and dialog boxes are provided for tasks such as adding mappable elements, working with ESQL, and working with submaps. Mappings that are created with the Message Mapping editor are automatically validated and compiled, ready for adding to a broker archive (BAR) file, and subsequent deployment to WebSphere Message Broker.

Message Mapping editor Source pane

Details of the elements present in the Source pane of the Message Mapping Editor.

The following example shows the “Message Mapping editor” on page 1530. The pane that is labelled as 1 in the example is the Source pane:



The following list describes the elements that are present in the Source pane:

- A source message is identified by $source.
- A source database is identified by $db:select.
- A database stored procedure is identified by $db:proc.
- A database user-defined function is identified by $db:func.
- A mapped entry is indicated by a blue triangle alongside the element. In this example, Customer_ID and Order_Date are mapped.
- Square brackets contain minimum and maximum occurrences of an element.
- An optional field is indicated by [0,1]. In this example, First_Class is optional.
- A repeating field is indicated by [minoccurs, maxoccurs].
- A choice field is indicated by a choice line; under the choice line are the possible choices. In this example, First_Class, Second_Class, and Airmail are choices of Delivery_Method.
- The type of each element is indicated in round brackets after the element name.
- If the message schema uses namespaces, the namespace prefix is shown before the element name, separated by a colon.

Use the Source pane to invoke a number of actions, a list of which is displayed when you right-click within the Source pane. The following table describes the available actions.

Action	Description	Related tasks
Undo	Undo previous action	
Redo	Redo previous action	
Revert	Discard	

Action	Description	Related tasks
Open Declaration (message)	Display the element definition from the message set. For this action to be available, select any source message element except LocalEnvironment or Headers.	
Open Declaration (database)	Display the database, schema, or table definition from the database. For this action to be available, select any source database object.	
Show Derived Types	Hide or display derived types for an element in the source or target pane. For this action to be available, select a target element displayed as a specialization folder in the source pane.	
Show Substituting elements	Hide or display the substituting elements of the head element in the source or target pane. For this action to be available, select a target element displayed as a substitutions folder in the source pane.	
Add Sources	Add a message definition or a database table to a source. For this action to be available, select any source object.	"Adding messages or message components to the source or target" on page 575, "Adding a database as a source or target" on page 575
Go To	For this action to be available, select any source object.	
Delete (message)	Remove a message and any existing maps from the source. For this action to be available, select the source message root ($source).	

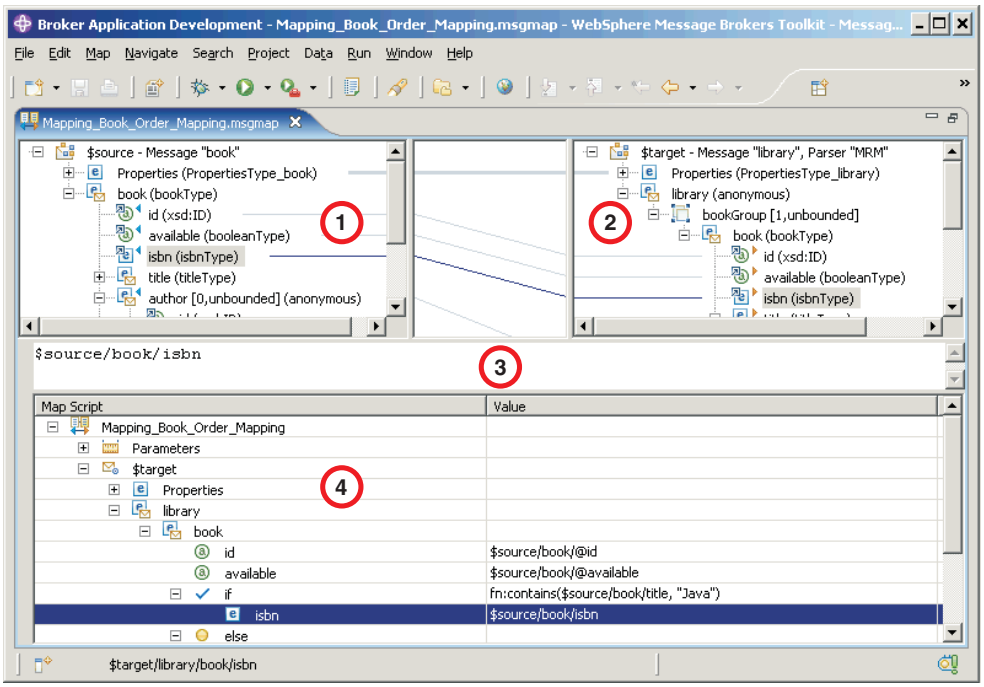
Action	Description	Related tasks
Delete (database)	Remove a database and any existing maps from the source. For this action to be available, select the source database root ($db:select).	
Delete (database stored procedure)	Remove a database stored procedure and any existing maps from the source. For this action to be available, select the database stored procedure root ($db:proc).	
Delete (database user-defined function)	Remove a database user-defined function and any existing maps from the source. For this action to be available, select the database user-defined function root ($db:func).	
Map from Source	Create a map between the focus source element and the focus target element. For this action to be available, select compatible source and target elements.	"Mapping a target element from source message elements" on page 564, "Mapping from source: by selection" on page 555
Map by Name	Create a map between the focus source element and the focus target element. For this action to be available, select compatible source and target elements.	"Mapping a target element from source message elements" on page 564, "Mapping from source: by name" on page 555
Accumulate	If the source and target fields contain numeric data types, this action maps all occurrences of a repeating source field to a non-repeating target, resulting in the sum of all the source elements. For this action to be available, select the source and target element.	"Configuring a repeating source and a non-repeating target" on page 571
Create New Submap	For this action to be available, select source and target elements that are either elements of complex types or wildcard elements.	"Creating and calling submaps and subroutines" on page 590, "Creating a new submap" on page 590, "Creating a new submap for a wildcard source" on page 591

Action	Description	Related tasks
Create New Database Submap	Create a submap to modify a database	"Creating a submap to modify a database" on page 592
Call Existing Submap	Call an existing submap	"Creating and calling submaps and subroutines" on page 590, "Calling a submap" on page 594
Call ESQL Routine	Call an ESQL routine	"Creating and calling submaps and subroutines" on page 590, "Calling an ESQL routine" on page 596
Call Java Method	Call a Java Method	"Calling a Java method" on page 597
Add or Remove Headers and Folders	Include message headers and folders for source messages in a message map	"Mapping headers and folders" on page 573
Toggle Add/Remove Stored Procedure Return Value	Specify if a database stored procedure sets a return value. DB2 on z/OS and Oracle stored procedures do not set a return value.	"Mapping a target element from database stored procedures" on page 585
Add or Remove Result Set Columns	Specify the Result Set Columns for a database stored procedure	"Mapping a target element from database stored procedures" on page 585
Save	Save the .msgmap file	

Message Mapping editor Target pane

Details of the elements present in the Target pane of the Message Mapping Editor.

The following example shows the "Message Mapping editor" on page 1530. The pane that is labelled as 2 in the example is the Target pane:



The following list describes the elements that are present in the Target pane:

- A target message is identified by $target.
- A mapped entry is indicated by a yellow triangle alongside the element. In this example, Customer_ID, Order_Number, and Order_Date are mapped.
- Square brackets contain minimum and maximum occurrences of an element.
- An optional field is indicated by [0,1]. In this example, First_Class is optional.
- A repeating field is indicated by [minoccurs, maxoccurs].
- A choice field is indicated by a choice line; under the choice line are the possible choices. In this example, First_Class, Second_Class, and Airmail are choices of Delivery_Method.
- The type of each element is indicated in round brackets after the element name.
- If the message schema uses namespaces, the namespace prefix is shown before the element name, separated by a colon.

Use the Target pane to invoke a number of actions, a list of which is displayed when you right-click within the Target pane. The following table describes the available actions.

Action	Description	Related tasks
Undo	Undo previous action	
Redo	Redo previous action	
Revert	Discard	

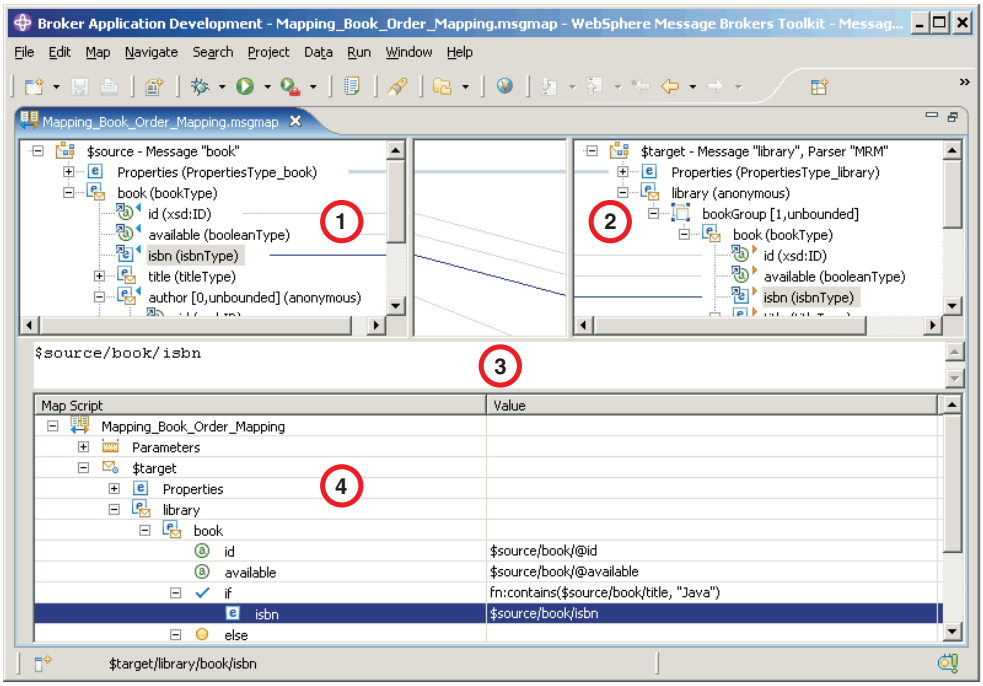
Action	Description	Related tasks
Open Declaration (message)	Display the element definition from the message set. For this action to be available, select any target message element except LocalEnvironment or Headers.	
Open Declaration (database)	Display the database, schema, or table definition from the database. For this action to be available, select any target database object.	
Show Derived Types	Hide or display derived types for an element in the source or target pane. For this action to be available, select a target element displayed as a specialization folder in the target pane.	
Show Substituting elements	Hide or display the substituting elements of the head element in the source or target pane. For this action to be available, select a target element displayed as a substitutions folder in the target pane.	
Add Sources and Targets	Add a message definition or a database table to a source. For this action to be available, select any target object.	"Adding messages or message components to the source or target" on page 575, "Adding a database as a source or target" on page 575
Go To	For this action to be available, select any target object.	
Delete (message)	Remove a message and any existing maps from the source. For this action to be available, select the target message root ($target).	

Action	Description	Related tasks
Map from Source	Create a map between the focus source element and the focus target element. For this action to be available, select compatible source and target elements.	"Mapping a target element from source message elements" on page 564, "Mapping from source: by selection" on page 555
Map by Name	Create a map between the focus source element and the focus target element. For this action to be available, select compatible source and target elements.	"Mapping a target element from source message elements" on page 564, "Mapping from source: by name" on page 555
Enter Expression	For this action to be available, select any target object except $target	"Setting the value of a target element to a constant" on page 566, "Setting the value of a target element using an expression or function" on page 568
Accumulate	If the source and target fields contain numeric data types, this action maps all occurrences of a repeating source field to a non-repeating target, resulting in the sum of all the source elements. For this action to be available, select the source and target element.	"Configuring a repeating source and a non-repeating target" on page 571
Create New Submap	For this action to be available, select source and target elements that are either elements of complex types or wildcard elements.	"Creating and calling submaps and subroutines" on page 590, "Creating a new submap" on page 590, "Creating a new submap for a wildcard source" on page 591
Call Existing Submap	Call an existing submap	"Creating and calling submaps and subroutines" on page 590, "Calling a submap" on page 594
Call ESQL Routine	Call an existing ESQL routine	"Creating and calling submaps and subroutines" on page 590, "Calling an ESQL routine" on page 596
Save	Save the .msgmap file	

Message Mapping editor Edit pane

Details of how you use the Edit pane of the Message Mapping Editor.

The following example shows the "Message Mapping editor" on page 1530. The pane that is labelled as 3 in the example is the Edit pane:



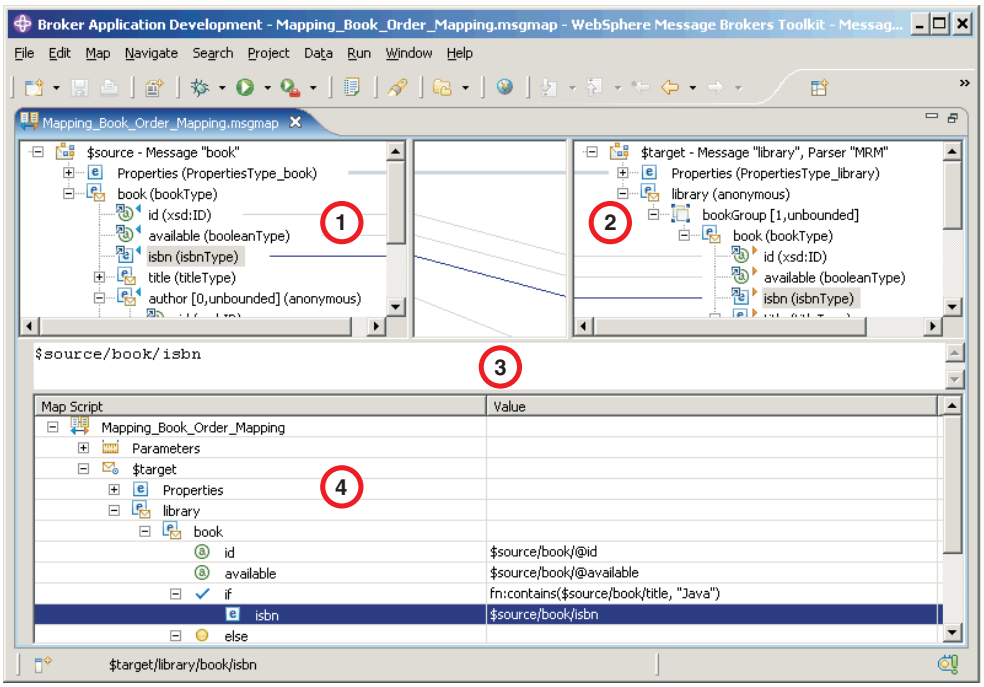
When you have selected a source or target element, use the Edit pane to enter an expression. Right-click inside the Edit pane to invoke a list of available actions, most of which are standard Windows functions, such as cut, copy, and paste. Click **Edit** → **Content Assist** (or press Ctrl+Space) to access ESQL Content Assist, which provides a drop-down list of functions that are available in a Mapping node.

To display the definition associated with a selected element or database object, right-click in the Edit pane, and click **Open Declaration**. The appropriate editor opens to display the definition associated with the element or database definition.

Message Mapping editor Spreadsheet pane

Details of the actions available to you in the Spreadsheet pane of the Message Mapping Editor.

The following example shows the “Message Mapping editor” on page 1530. The pane that is labelled as 4 in the example is the Spreadsheet pane:



Use the Spreadsheet pane to invoke a number of actions, a list of which is displayed when you right-click within the Spreadsheet pane. The following table describes the available actions.

Action	Description	Related tasks
Undo	Undo previous action	
Redo	Redo previous action	
Revert	Discard	
Open Declaration (message)	Display the element definition from the message set. For this action to be available, select any message element except LocalEnvironment or Headers.	
Open Declaration (database)	Display the database, schema, or table definition from the database. For this action to be available, select any database object.	
Add Sources and Targets	Add a message definition to a target.	"Adding messages or message components to the source or target" on page 575, "Adding a database as a source or target" on page 575
Copy	Copy the selected item to the clipboard.	

Action	Description	Related tasks
Paste	Paste the item from the clipboard.	
Delete	Remove a row from the Spreadsheet.	
For	Define a repeating condition.	"Configuring a repeating source and a non-repeating target" on page 571, "Configuring a repeating source and a repeating target" on page 572
If	Define what must evaluate to 'true' to process subsequent mappings.	"Configuring a repeating source and a non-repeating target" on page 571, "Configuring conditional mappings" on page 570
ElseIf	Define what must evaluate to 'true' to process subsequent mappings if previous If or Elseif does not evaluate to 'true'..	"Configuring a repeating source and a non-repeating target" on page 571, "Configuring conditional mappings" on page 570
Else	Placeholder to process subsequent mappings if previous If or ElseIf does not evaluate to 'true'.	"Configuring conditional mappings" on page 570
Select Data Source	Define a database to be used in the mapping.	
Insert Children	Expand a structure so that each of its children has a row in the spreadsheet.	
Insert Sibling After	Create a number of new rows in the spreadsheet to set the values of specific instances of a repeating field. Can also be used to insert any non-repeating element, attribute or database column if valid at the selected location.	"Configuring a non-repeating source and a repeating target" on page 572
Insert Sibling Before	Create a number of new rows in the spreadsheet to set the values of specific instances of a repeating field. Can also be used to insert any non-repeating element, attribute or database column if valid at the selected location.	"Configuring a non-repeating source and a repeating target" on page 572
Replace	Substitute an element, attribute or database column in the spreadsheet with a similar item, retaining the mapping expression and any child mapping statements.	
Save	Save the .msgmap file.	

Mapping node

The Mapping node has one or more mappings that are stored in message map files (with a `.msgmap` file extension). These files are configured using the "Message Mapping editor" on page 1530.

A Mapping node must contain the following inputs and outputs:

- Zero or one source (input) messages
- Zero or more source (input) databases
- One or more target (output) messages

You must define, in message definition files in a message set, the source and target messages that are to be mapped. You can specify the parser of the source message at run time (for example, in an MQRFH2 header), but the target message is built using the runtime parser that is specified by the *Message Domain* property of the message set.

If a message mapping is between elements of different types, you might need to include casts in your mapping definitions, depending on which runtime parser is specified by the *Message Domain* property of your message set.

The Mapping node uses a language to manipulate messages that are based on XPath.

To develop message mappings for a Mapping node, use the Message Mapping editor, which provides separate panes for working with sources, targets and expressions.

Mapping node syntax

In a Mapping node, the source message, if present, is identified in the "Message Mapping editor" on page 1530 by $source.

The message tree is represented in XPath format. For example, if you have an element called Body within a source message called Envelope, this is represented in the Mapping node as:

```
$source/soap11:Envelope/soap11:Body
```

where *soap11* is a namespace prefix.

The first target message is identified by $target; additional target messages are identified by $target_1, $target_2, and so on.

The first source database is identified by $db:select; additional source databases are identified by $db:select_1, $db:select_2, and so on.

The first source database stored procedure is identified by $db:proc; additional source stored procedures are identified by $db:proc_1, $db:proc_2, and so on.

The first source database user-defined function is identified by $db:func; additional source user-defined functions are identified by $db:func_1, $db:func_2, and so on.

The database element is represented in the following format:

```
$db:select.DB.SCH.TAB.COL1
```

where:

DB is the database name

SCH is the database schema name

TAB is the table name

COL1 is the column name

You can also use the Mapping node to:

- make comparisons
- perform arithmetic
- create complex conditions

The comparison operators are:

= equals

!= not equals

> greater than

>= greater than or equals

< less than

<= less than or equals

The arithmetic operators are:

+ plus

- minus

* multiply

div divide

Conditional operators ‘or' and ‘and' are supported (these are case-sensitive).

The following objects can be mapped:

- Local Environment
  - Destination
  - WrittenDestination
  - File
  - SOAP
  - TCPIP
  - ServiceRegistry
  - Adapter
  - Wildcard
  - Variables
- Message headers (optional)
  - MQ Headers
  - HTTP Headers
  - JMSTransport
  - E-mail Headers
- Message elements
- Database columns

Database objects with names that do not conform to the XML NCName format

Some database objects have names that do not conform to the XML NCName format (for example, the names contains characters like '#', or '$'). To reference such database objects use the `msgmap:db-path` function. For more information, see "Predefined mapping functions" on page 1551

Mapping node functions

You can configure your message mappings to use a variety of predefined and user-defined functions.

The following predefined functions are available to use in your message maps:

- ESQL - prefixed esql:
- XPath - prefixed fn:
- Mapping - prefixed msgmap:
- Schema casts - prefixed xs:

Not all ESQL functions can be used in a Mapping node. For information about which functions are supported, and for a description of how to achieve equivalent processing for ESQL functions that are not supported, see the ESQL topics. For information about the predefined ESQL functions, see "Predefined ESQL mapping functions."

The fn:true() function (which always returns true) and the fn:false() function (which always returns false) are examples of XPath functions. You can get more information about the other XPath functions and XPath syntax from the online W3C XML Path Language document. For information about the predefined XPath functions, see "Predefined XPath mapping functions" on page 1547.

For information about the predefined mapping functions, see "Predefined mapping functions" on page 1551. See "Mapping node casts" on page 1553 for a list of the schema casts.

The Mapping node can also:

- Set the value of a target to a WebSphere MQ constant. The expression to set the value looks similar to a function with $mq: used as the prefix.
- Call a Java method directly. The expression to set the value looks similar to a function with java: used as a prefix.

Predefined ESQL mapping functions:

A table of predefined ESQL functions for use with message maps.

This table details the predefined ESQL mapping functions that are available to use with message maps:

Name	ESQL equivalent	Notes
Numeric functions: abs absval acos asin atan atan2 bitand bitnot bitor bitxor ceil ceiling cos cosh cot degrees exp floor in log log10 mod power radians rand sign sin sinh sqrt tan tanh truncate	ESQL function of the name same name such as ABS and ABSVAL.	The same parameters apply as for ESQL.

Name	ESQL equivalent	Notes
String functions: left length lower lcase ltrim replace replicate right rtrim space translate upper ucase	ESQL function of the same name such as LEFT and LENGTH.	The same parameters apply as for ESQL.
Field functions: bitstream fieldname fieldnamespace fieldtype fieldvalue lastmove samefield	ESQL function of the same name such as BITSTREAM and FIELDNAME.	The same parameters apply as for ESQL.
asbitstream	These signatures are supported: ASBITSTREAM(*FieldRef*) ASBITSTREAM(*FieldRef*, *typeExp*, *setExp*, *formatExp*) ASBITSTREAM(*FieldRef*, *typeExp*, *setExp*, *formatExp*, *encodingExp*, *ccsidExp*) ASBITSTREAM(*FieldRef*, *typeExp*, *setExp*, *formatExp*, *encodingExp*, *ccsidExp*, *options*)	*FieldRef* is a source field reference such as $source/po:PurchaseOrder *typeExp* is a string literal of the name of the message body, such as purchaseOrder, optionally qualified with a namespace URI, such as {http://www.ibm.com}:purchaseOrder *setExp* is a string literal of the name of the message set, such as PurchaseOrder *formatExp* is a string literal of the wire format of the message, such as XML1 *encodingExp* and *ccsidExp* evaluate to integers with values corresponding to ESQL ENCODING and CCSID constants. *options* is an ESQL constant or bit-or of ESQL constant that evaluates to an integer.
cardinality	CARDINALITY	The same parameters apply as for ESQL.
coalesce	COALESCE	The same parameters apply as for ESQL.
current-date	CURRENT_DATE	No parameters apply.
current-gmtdate	CURRENT_GMTDATE	No parameters apply.
current-gmttime	CURRENT_GMTTIME	No parameters apply.
current-gmttimestamp	CURRENT_ GMTTIMESTAMP	No parameters apply.
current-time	CURRENT_TIME	No parameters apply.
current-timestamp	CURRENT_TIMESTAMP	No parameters apply.
date	DATE	
for	FOR (expression)	Optional parameters are not supported.
gmttime	GMTTIME	
gmttimestamp	GMTTIMESTAMP	

interval-year	INTERVAL YEAR	The same parameters apply as for ESQL. Some examples:
		
		`esql:interval-minute('90')`
		`esql:interval-year-to-month('1-06')`

interval-year-to-month	INTERVAL YEAR TO MONTH	
interval-month	INTERVAL MONTH	
interval-day	INTERVAL DAY	
interval-day-to-hour	INTERVAL DAY TO HOUR	
interval-day-to-minute	INTERVAL DAY TO MINUTE	
interval-day-to-second	INTERVAL DAY TO SECOND	
interval-hour	INTERVAL HOUR	
interval-hour-to-minute	INTERVAL HOUR TO MINUTE	
interval-hour-to-second	INTERVAL HOUR TO SECOND	
interval-minute	INTERVAL MINUTE	
interval-minute-to-second	INTERVAL MINUTE TO SECOND	
interval-second	INTERVAL SECOND	

is-null	Operand IS NULL	Some examples:
		
		`esql:is-null($source/po:purchaseOrder/po:comment)`
		`esql:is-null ($db:select.ACME.PARTS.INVENTORY.LAST_TRANSACTION)`

Name	ESQL equivalent	Notes
like	source LIKE pattern	For example: `esql:like ($source/po:purchaseOrder/shipTo/first_name,'Fred')`
	source LIKE pattern ESCAPE EscapeChar	For example: `esql:like ($source/po:purchaseOrder/shipTo /zip,'L6F$_1C7','$')`
local-timezone	LOCAL_TIMEZONE	
nullif	NULLIF	The same parameters apply as for ESQL.
overlay	OVERLAY Str1 PLACING Str2 FROM Start	For example: `esql:overlay ($source/po:purchaseOrder/shipTo/city,'abc',2)`
	OVERLAY Str1 PLACING Str2 FROM Start For Length	For example: `esql:overlay ($source/po:purchaseOrder/shipTo/city,'abcde',2,3)`
position	POSITION searchExp IN SourceExp	For example: `esql:position ('aet',$source/po:purchaseOrder/shipTo/first_name)`
	POSITION searchExp IN SourceExp FROM FromExp	For example: `esql:position ('do',$source/po:purchaseOrder/shipTo/last_name,1)`
	POSITION searchExp IN SourceExp FROM FromExp REPEAT RepeatExp	For example: `esql:position ('a',$source/po:purchaseOrder/billTo /first_name,1,2)`
round	ROUND	Optional parameters are not supported.
sqlcode	SQLCODE	No parameters apply.
sqlerrortext	SQLERRORTEXT	
sqlnativeerror	SQLNATIVEERROR	
sqlstate	SQLSTATE	
time	TIME	
timestamp	TIMESTAMP	The same parameters apply as for ESQL. For example: `esql:gmttimestamp ('1999-12-31 23:59:59.999999')`
trim-leading	TRIM LEADING FROM Source	For example: `esql:trim-leading ($source/po:purchaseOrder/shipTo/state)`
	TRIM LEADING Singleton FROM Source	For example: `esql:trim-leading ('G',$source/po:purchaseOrder/shipTo/zip)`
trim-trailing	TRIM TRAILING FROM Source	For example: `esql:trim-trailing ($source/po:purchaseOrder/billTo/last_name)`
	TRIM TRAILING Singleton FROM Source	For example: `esql:trim-trailing ('e',$source/po:purchaseOrder/billTo/street)`

Name	ESQL equivalent	Notes
trim-both	TRIM BOTH FROM Source	For example: `esql:trim-both ($source/po:purchaseOrder/shipTo/city)`
	TRIM BOTH Singleton FROM Source	For example: `esql:trim-both (",$source/po:purchaseOrder/shipTo/city)`
trim	TRIM Source	For example: `esql:trim ($source/po:purchaseOrder/shipTo/city)`
	TRIM Singleton FROM Source	For example: `esql:trim (",$source/po:purchaseOrder/shipTo/city)`
uuidasblob	UUIDASBLOB	Takes zero or more parameters as in ESQL.
uuidaschar	UUIDASCHAR	

Predefined XPath mapping functions:

A table of predefined XPath functions for use with message maps.

This table details the predefined XPath functions that are available for use with message maps. You can get more information about XPath functions and XPath syntax from the online W3C XML Path Language document.

Name	Parameters	Notes
true		
false		
sum	Source field from the message or database or “XPath "for expression"” on page 1550.	Source supports an XPath predicate. An XPath predicate is an expression enclosed in square brackets, serving to filter a sequence, retaining some items and discarding others. Predicates are supported on the XPath aggregate functions of `avg`, `count`, `max`, `min`, and `sum`. Predicates must be in one of the two forms: • An integer literal – aggregation is done for elements whose context position equals the integer. • A Boolean expression – aggregation is done for elements that make the Boolean expression evaluate to true. See “Aggregating XPath expressions conditionally” on page 1548 for more information.
avg		
max		
min		
count		
concat	Two or more strings.	You cannot use `fn:concat($source/myElem)` to concatenate instances of 'myElem'.
not	1- Expression resolved to a Boolean value.	
exists	Source field from the message or database.	
empty		

Name	Parameters	Notes
substring	1- String 2- Zero-bases starting index 3- Length	For example: `fn:substring ($source/po:purchaseOrder/billTo/street, 3, 5)`
substring-before	Two strings	Returns the substring of the first string that precedes the first occurrence of the second string.
substring-after	Two strings	Returns the substring of the first string that follows the first occurrence of the second string.
starts-with	Two strings	Returns a Boolean value indicating whether the first string starts with the second string.
ends-with	Two strings	Returns a Boolean value indicating whether the first string ends with the second string.
contains	Two strings	Returns a Boolean value indicating whether the first string contains the second string.
year-from-dateTime	1- xs:dateTime	For example: `fn:month-from-dateTime (xs:dateTime($source/po:purchaseOrder /shipTo/datetime))` where `$source/po:purchaseOrder/shipTo/datetime` is xs:string.
month-from-dateTime		
day-from-dateTime		
hours-from-dateTime		
minutes-from-dateTime		
seconds-from-dateTime		
year-from-date	1-xs:date	For example: `fn:year-from-date(xs:date ($source/po:purchaseOrder/billTo/date))` where `$source/po:purchaseOrder/billTo/date` is xs:string.
month-from-date		
day-from-date		

hours-from-time	1- xs:time	Some examples:
		
		`fn:hours-from-time(xs:time("13:20:10:5"))`
		`fn:hours-from-time(xs:time ($source/po:purchaseOrder/shipTo/time))`

minutes-from-time		
seconds-from-time		
years-from-duration	1- xdt:dayTimeDuration	For example: `fn:minutes-from-duration (xdt:dayTimeDuration(PT47H30M))`
months-from-duration		
days-from-duration		
hours-from-duration		
minutes-from-duration		
seconds-from-duration		

Aggregating XPath expressions conditionally:

You can perform calculations using conditions, called predicates in XPath, on the aggregate functions in the Mapping node. The aggregate functions are `avg`, `count`, `max`, `min`, and `sum`.

XPath aggregate functions

XPath aggregate functions take a sequence as their argument:

```
fn:count($arg as item()*) as xs:integer
fn:avg($arg as xs:anyAtomicType*) as xs:anyAtomicType?
fn:max($arg as xs:anyAtomicType*) as xs:anyAtomicType?
fn:min($arg as xs:anyAtomicType*) as xs:anyAtomicType?
fn:sum($arg as xs:anyAtomicType*) as xs:anyAtomicType?
```

where the atomic value can be, for example, an integer, a string, or a Boolean value.

In an aggregation

```
fn:sum($source/inventory/category/product/price)
```

the node

```
$source/inventory/category/product/price
```

is put into its typed value, which is a sequence of `Atomictype` `price` values.

In the following aggregation, the argument is a node `tns:price`, which is contained in `tns:product`, which is in turn contained in `tns:category`, and so on.

```
fn:sum($source/tns:Inventory/tns:category/tns:product/tns:price)
```

How aggregation works in this sequence depends upon the scope of the iteration (or `for`) loop for `product` and `category`. For example, if there is no iteration loop, summation is done for all instances of all `prices` in all `products` in all `categories`.

However, if there is an iteration loop on `category`, one summation is calculated for each `category`. The summation is obtained from all `prices` of all `products` of a particular `category` being iterated upon.

You can add a predicate (see predicates for further information) to make the result more specific. For example:

```
fn:sum($source/tns:Inventory/tns:category[$source/tns:Inventory/tns:category/
  tns:c_id='abc' ]/tns:product[3]/tns:price)
```

has a predicate of `tns:c_id='abc'` on `category`, and a predicate of 3 on `product`. The result that you obtain is the third `product` in the `category` that has `tns:c_id='abc'`.

In both of the preceding examples, whether the expression contains a condition or not, you reference one source item only; it is `price` that is aggregated.

Predicates are supported only when they are used in message sources. For example, the `[ boolean expression ]` must be used within a segment of a path that represents a message source. Predicates are not supported in `$db:select` or `$db:proc`.

Consider a more complicated scenario, where product prices are kept in a database table, and the input message contains the quantity of the product being ordered. The objective is to calculate a total price by adding up all prices multiplied by quantity.

It might seem to be straightforward to write the following aggregation function:

```
fn:sum($source/inventory/product/quantity × $db:select/PRICE_TB/PRICE)
```

However, the first step of evaluating an XPath arithmetic expression is to evaluate its operands. If any operand is evaluated into a sequence of more than one item,

either only the first item is used in the arithmetic expression, or an error is raised; for further information, see Arithmetic expressions.

You, therefore, cannot add up the product of `quantity` and `PRICE` for each `product`. If you want to aggregate the result of an arithmetic expression, see "XPath "for expression"."

XPath "for expression":

You can use the "for expression" to perform specific calculations as the argument of an aggregation function in the Mapping node.

For expressions

To perform a specific operation you can use the XPath `for expression` iteration facility; see for expression for further information.

You can express a sequence of price multiplied by quantity in many ways using the `for expression`. For example:

- `for $i in $source/inventory/category/product return $i/price * $i/quantity`

  You can use a similar expression when all operands of the return expression (`price` and `quantity`) are defined in the same repeatable container (`product`). Only one variable is needed in the `for expression`.
- ```
  for $i in $db:select, $j in $source/inventory/category/product
  [$i.db1.sch2.PRICE_TB.PROD_ID=$j/product_id]
  return $i.db1.sch2.PRICE_TB.PRICE * $j/quantity
  ```

  You can use a similar expression when some operands of the return expression (`price`) are kept in a database table, and other operands (`quantity`) are kept in a message. At least one variable is needed for the path to a database result set, and another variable is needed for the path to a repeatable XML element.
- `for $i in $source/order_info/product1, $j in $source/price_record/product2 [$i/product_id=$j/product_id] return $i/quantity * $j/price`

  You can use a similar expression when the operands (`price` and `quantity`) are defined in different repeatable elements (`product1` and `product2`) in the source message.

When you have an expression that represents a sequence of items, aggregation can be done by using the `for expression` as the argument of the aggregation function.

Examples

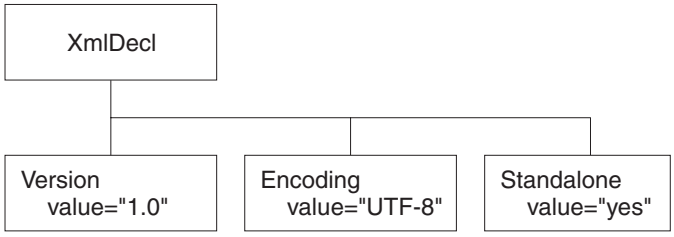
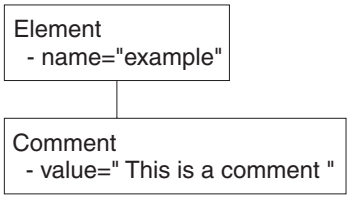
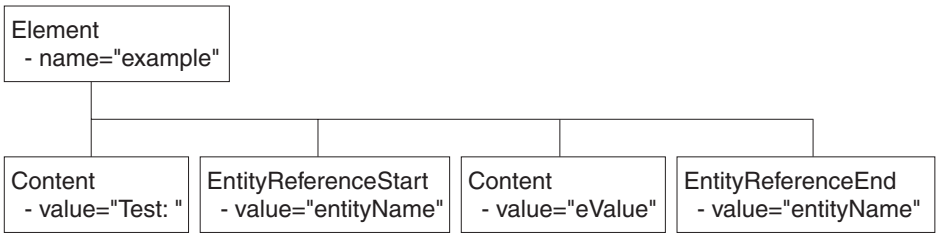
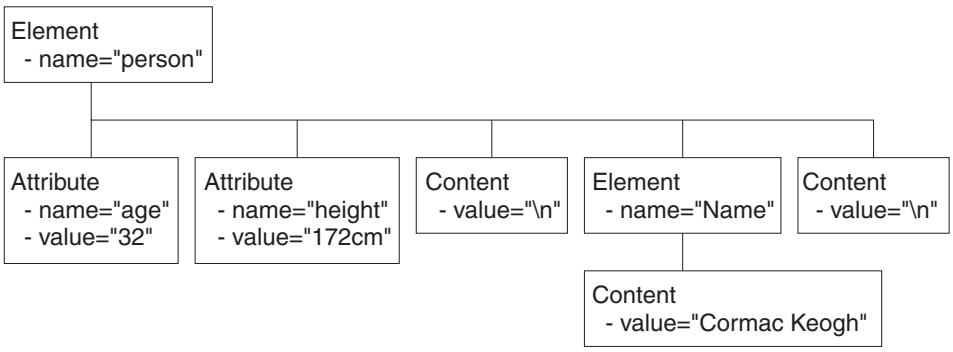
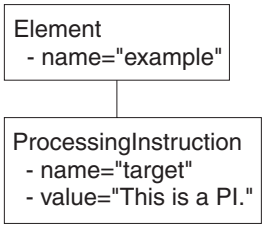
Obtain the average cost - that is, price multiplied by quantity - for all products:

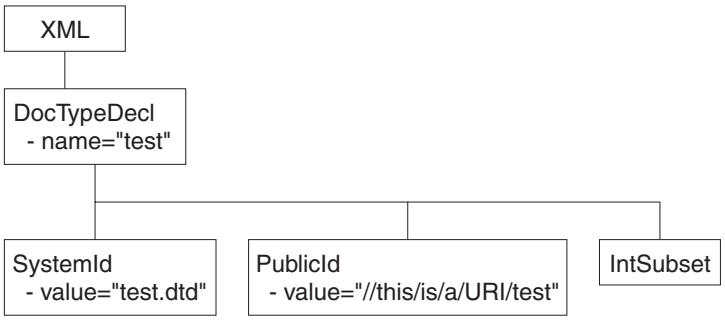
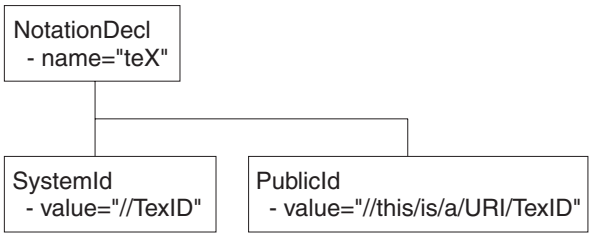
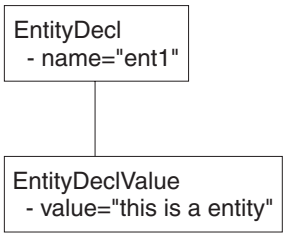
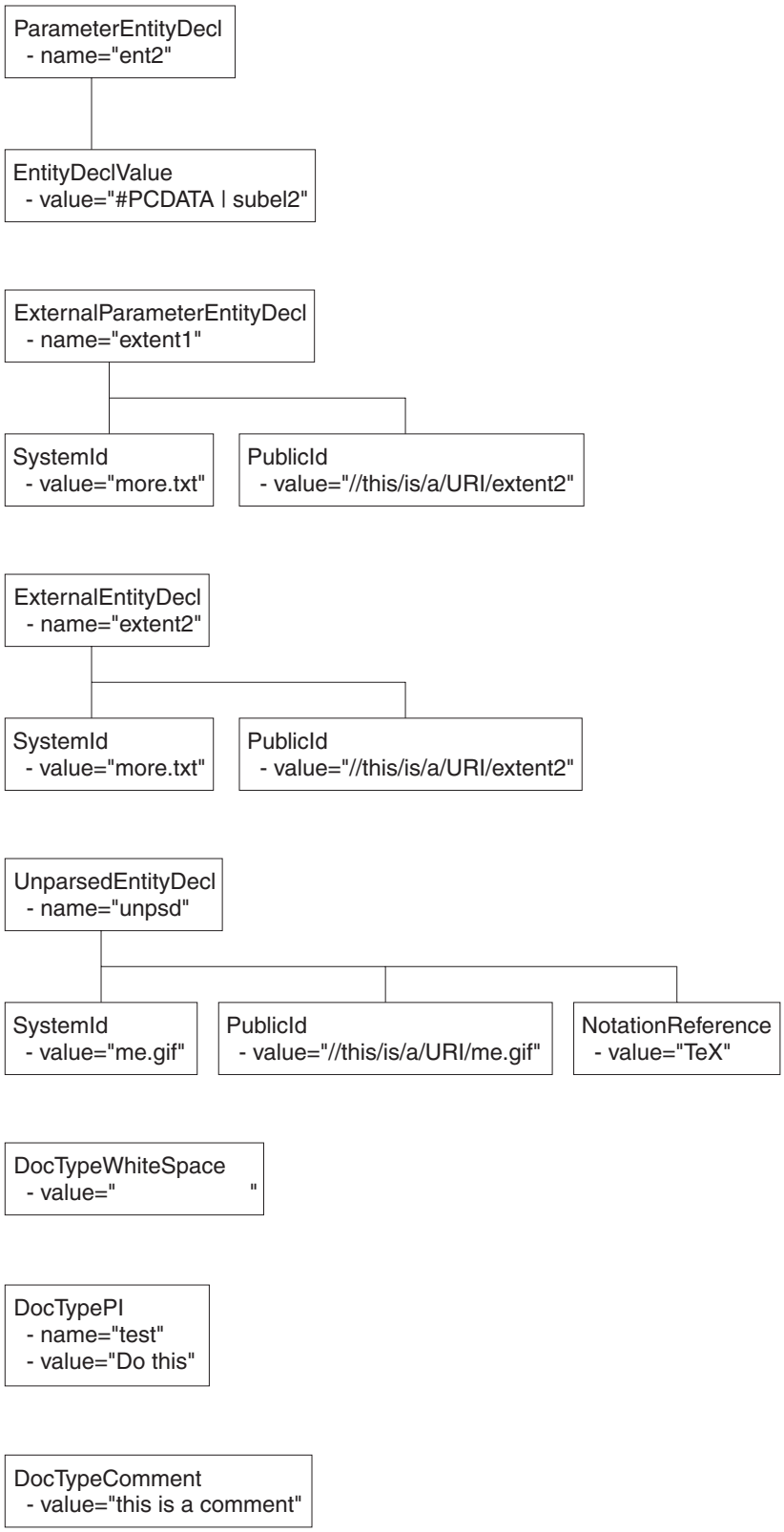
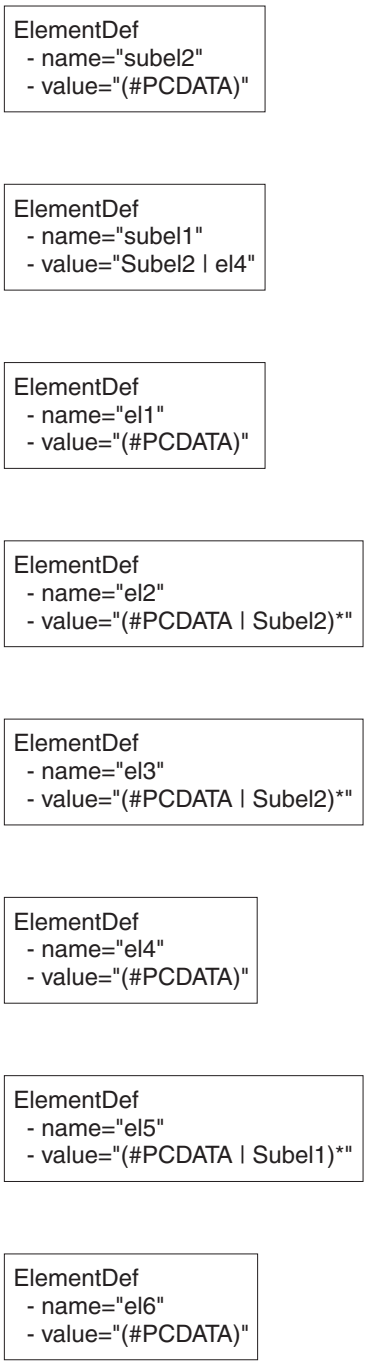
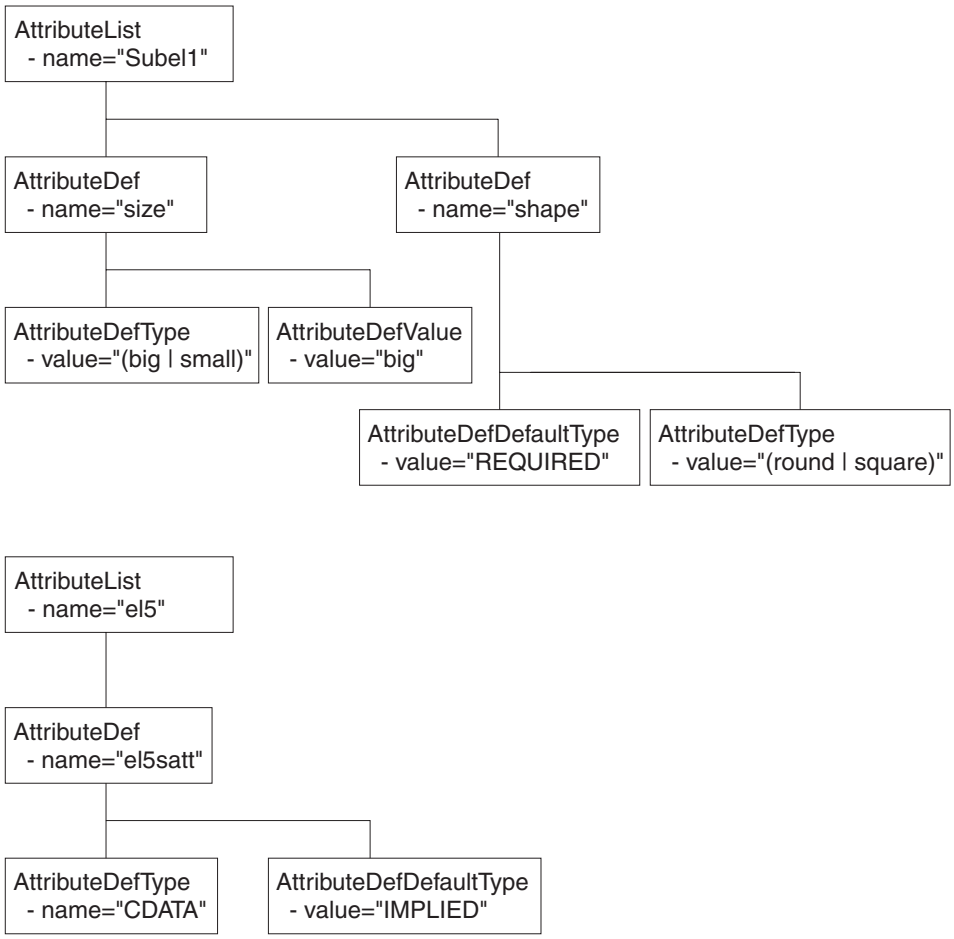
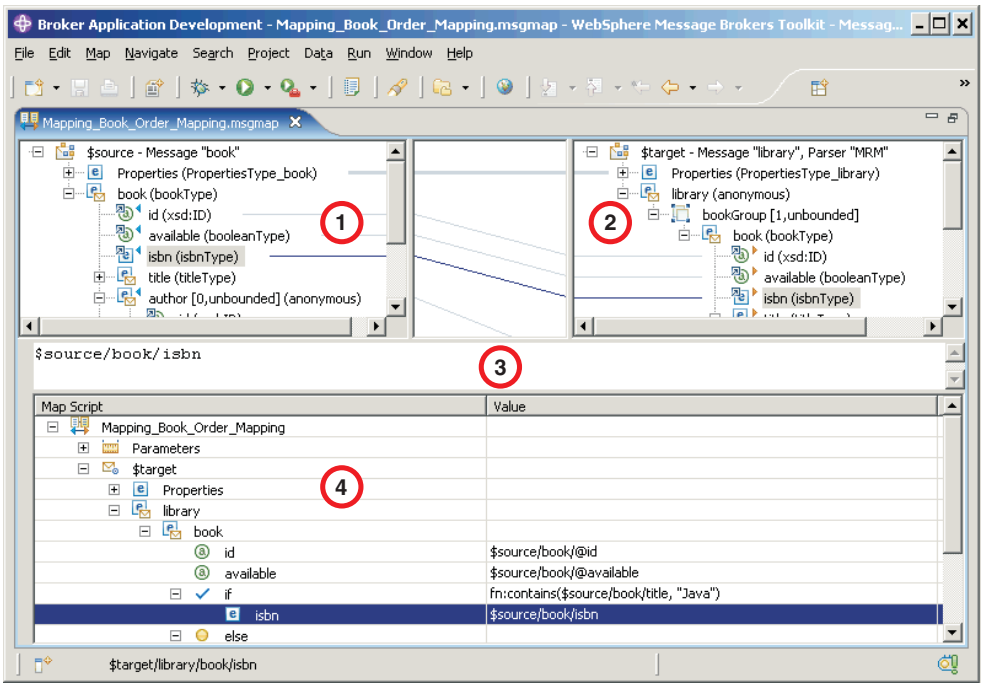
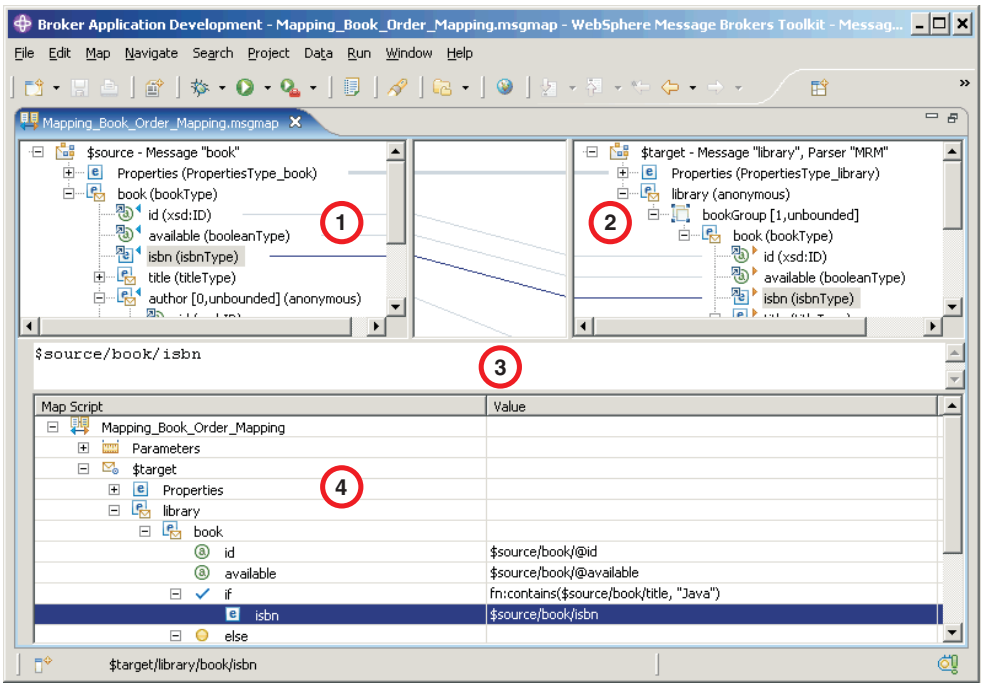
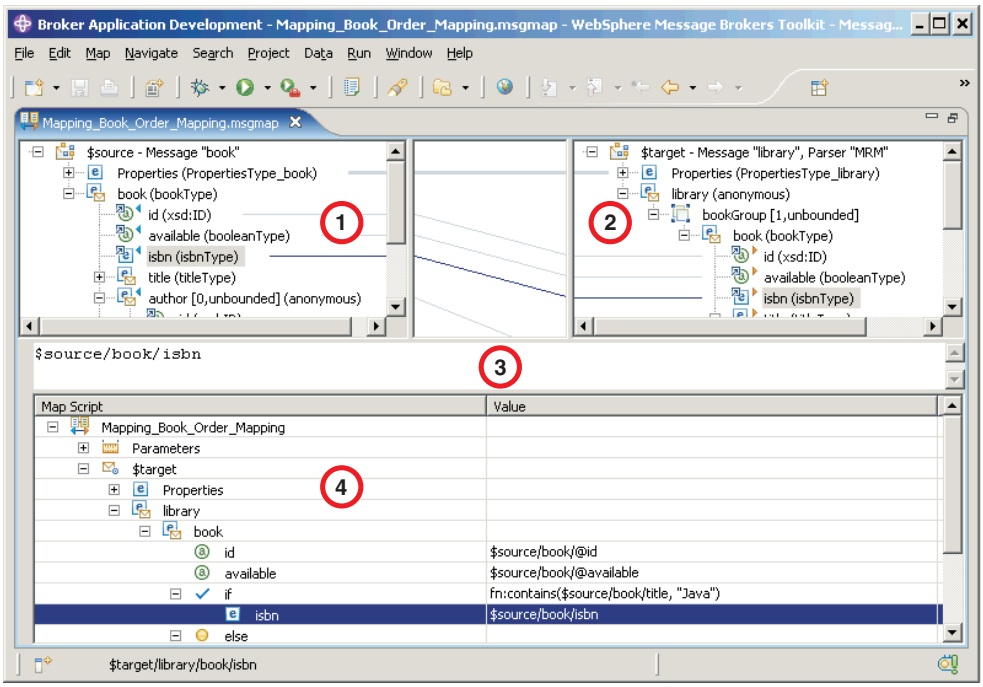
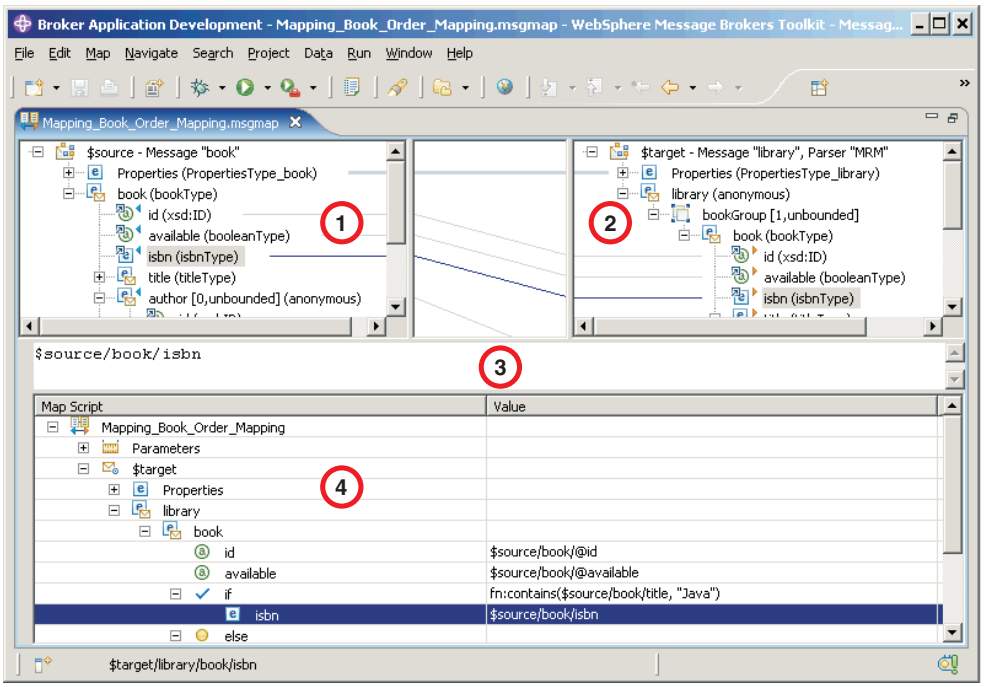
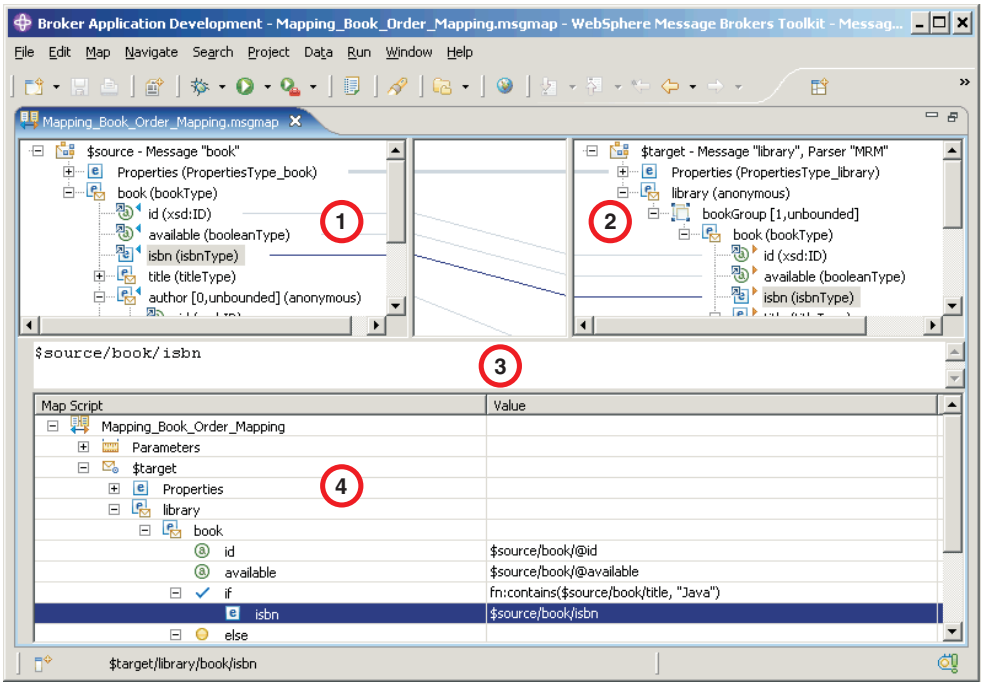
```
fn:avg(for $i in $source/inventory/category/product
   return $i/price * $i/quantity)
```

Obtain the total cost of all products where prices are from a database, and quantities are from a message and paired up based on product identifier:

```
fn:sum(for $i in $db:select, $j in $source/inventory/category/product
   [$i.db1.sch2.PRICE_TB.PROD_ID=$j/product_id]
   return $i.db1.sch2.PRICE_TB.PRICE * $j/quantity)
```

Obtain the length of the longest product identifier text:

```
fn:max(for $i in $source/inventory/category/product
   return esql:length($i/id))
```
```

Obtain the minimum price when the price was stored in a message as a string, and convert each source item into a numeric value before using an aggregate function:

```
fn:min(for $i in $source/inventory/category/product
   return xs:decimal($i/price))
```

The aggregation functions that are supported in WebSphere Message Broker Version 6.1.0.2, and earlier, can be expressed using a `for expression`.

For example, in WebSphere Message Broker Version 6.1.0.2 you can have:

```
fn:sum($source/inventory/category/product/price)
```

and this format is still supported. This expression is equivalent to:

```
fn:sum(for $i in $source/inventory/category/product
   return $i/price)
```

Predefined mapping functions:

Some predefined mapping functions are provided for use with message maps.

This table details the predefined mapping functions that are available to use with message maps.

Name	Parameters	Return	Notes
cdata-element	One string	Nothing	Create an XML element with CData content in the following target message domains: • XMLNSC • SOAP • XMLNS • XML • JMSMap • JMSStream For example: `msgmap:cdata-element('<date><month>05 </month><day>11</day><year>2008</year></date>')`

db-path	These signatures are supported:
	msgmap:db-path(*databasePath*)
	msgmap:db-path(*databasePath*, *delimiter*)
		Nothing	Used when the database path does not conform to the XML NCName format. For example, the database path contains characters like '#', or '$'.
			databasePath is a reference to either a `$db:select` statement, a `$db:insert` statement, a `$db:update` statement, a `$db:delete` statement, a `$db:proc` statement, or a `$db:func` statement.
			delimiter is a string literal used between the segments in the database path. *delimiter* has a default value of '.'.
			For example:
			`msgmap:db-path("$db:select.DB#1.SCH$#1. CustInfo.Name")`
			In this case, the default value, '.', is used as the delimiter.
			`msgmap:db-path("$db:select/DB#1/SCH$#1/ Cust.Info/Name", "/")`
			In this case, '/' is used as the delimiter because the database table name includes the default value.

Name	Parameters	Return	Notes

occurrence	Source field from the message or database	The index of the source field.	Often used in a condition statement when source repeats to execute specific statements for a specific occurrence. For example:
			`msgmap:occurrence ($source/po:purchaseOrder /items)=2`
			
			means the second field,
			`po:purchaseOrder`
			
			is being processed. Use code similar to the previous example if you want to specify a particular source occurrence. If you only want to assign the index value to a target you can specify the following code:
			`msgmap:occurrence ($source/po:purchaseOrder /items)`

exact-type	1- Source field from the message or database
	2- Namespace prefix
	3- Name of the type
		True if the source is of the specified type in the specified namespace.	Often used in a condition to execute specific statements for a specific source type. For example:
			`msgmap:exact-type ($source/tn1:msg2,'tn1', 'extendedMsgType')`
			
			The namespace prefix can be '*', indicating that only the name of the type is to be checked. For example:
			`msgmap:exact-type($source/tn1:msg2,'*', 'extendedMsgType')`
			
			returns true if the element
			`$source/tn1:msg2`
			
			has a type of
			`extendedMsgType`
			
			in any namespace.

empty-element()	None	Nothing	Creates an XML element with an empty tag. For example, if an element is named MyElement and the mapping expression for MyElement is set to msgmap:empty-element(), the output message will contain an element with no content: `<MyElement/>` Call this function only for an XML element.

Name	Parameters	Return	Notes
element-from-bitstream	These signatures are supported: msgmap:element-from-bitstream(*StreamRef*) msgmap:element-from-bitstream(*StreamRef*, *typeExp*, *setExp*, *formatExp*) msgmap:element-from-bitstream(*StreamRef*, *typeExp*, *setExp*, *formatExp*, *encodingExp*, *ccsidExp*) msgmap:element-from-bitstream(*StreamRef*, *typeExp*, *setExp*, *formatExp*, *encodingExp*, *ccsidExp*, *options*)	Nothing	Used to parse a bit stream. This function can be called only for a message element target. The parsed bit stream is placed in the target message tree as the target element. *StreamRef* is a reference to a BLOB of stream, such as $source/BLOB or $db:select.dsn.schema.table.column *typeExp* is a string literal of the name of the message body, such as purchaseOrder, optionally qualified with a namespace URI, such as {http://www.ibm.com}:purchaseOrder *setExp* is a string literal of the name of the message set, such as PurchaseOrder *formatExp* is a string literal of the wire format of the message, such as XML1 *encodingExp* and *ccsidExp* evaluate to integers with values corresponding to ESQL ENCODING and CCSID constants *options* is an ESQL constant or bit-or of ESQL constants that evaluate to an integer.

Mapping node casts

Source and target elements can be of different types in a Mapping node.

Depending on which runtime parsers are used, automatic casting cannot be done. In these cases, use one of the cast functions shown in the following table.

Function name	Signature	Parameter type
xs:anyURI	xs:anyURI(*&exp*) xs:anyURI(*&exp*, *&ccsid*) xs:anyURI(*&exp*, *&ccsid*, *&encoding*)	*$exp* is a reference to a source element of one of the following types: • xs:base64Binary • xs:boolean • xs:date • xs:dateTime • xs:dayTimeDuration • xs:decimal • xs:double • xs:duration • xs:float • xs:gDay • xs:gMonth • xs:gMonthDay • xs:gYear • xs:gYearMonth • xs:hexBinary • xs:int • xs:integer • xs:long • xs:QName • xs:string • xs:time • xs:yearMonthDuration *&ccsid* is one of the following values: • $source/Properties/CodedCharSetId • $source/MQMD/CodedCharSetId • An integer. For example, 1208 for UTF-8 *&encoding* is one of the following values: • $source/Properties/Encoding • $source/MQMD/Encoding • $mq:MQENC_WINDOWS (546) • $mq:MQENC_UNIX (273) • $mq:MQENC_390 (785) • Other $mq:MQENC_* constants and their BIT OR

Function name	Signature	Parameter type
xs:base64Binary	xs:base64Binary(*&exp*) xs:base64Binary(*&exp*, *&ccsid*) xs:base64Binary(*&exp*, *&ccsid*, *&encoding*)	*$exp* is a reference to a source element of one of the following types: • xs:hexBinary • xs:int • xs:string *&ccsid* is one of the following values: • $source/Properties/CodedCharSetId • $source/MQMD/CodedCharSetId • An integer. For example, 1208 for UTF-8 *&encoding* is one of the following values: • $source/Properties/Encoding • $source/MQMD/Encoding • $mq:MQENC_WINDOWS (546) • $mq:MQENC_UNIX (273) • $mq:MQENC_390 (785) • Other $mq:MQENC_* constants and their BIT OR
xs:boolean	xs:boolean(*&exp*)	*$exp* is a reference to a source element of type xs:string.
xs:date	xs:date(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:dateTime • xs:gDay • xs:gMonth • xs:gMonthDay • xs:gYear • xs:gYearMonth • xs:string
xs:dateTime	xs:dateTime(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:date • xs:string • xs:time
xs:dayTimeDuration	xs:dayTimeDuration(*&exp*)	*$exp* is a reference to a source element of type xs:string.

Function name	Signature	Parameter type
xs:decimal	xs:decimal(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:decimal • xs:float • xs:int • xs:integer • xs:string
xs:double	xs:double(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:duration • xs:float • xs:int • xs:string
xs:duration	xs:duration(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:decimal • xs:float • xs:int • xs:string
xs:float	xs:float(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:double • xs:duration • xs:int • xs:string
xs:gDay	xs:gDay(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:date • xs:dateTime • xs:gMonth • xs:gMonthDay • xs:gYear • xs:gYearMonth • xs:string

Function name	Signature	Parameter type
xs:gMonth	xs:gMonth(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:date • xs:dateTime • xs:gDay • xs:gMonthDay • xs:gYear • xs:gYearMonth • xs:string
xs:gMonthDay	xs:gMonthDay(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:date • xs:dateTime • xs:gDay • xs:gMonth • xs:gYear • xs:gYearMonth • xs:string
xs:gYear	xs:gYear(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:date • xs:dateTime • xs:gDay • xs:gMonth • xs:gMonthDay • xs:gYearMonth • xs:string
xs:gYearMonth	xs:gYearMonth(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:date • xs:dateTime • xs:gDay • xs:gMonth • xs:gMonthDay • xs:gYear • xs:string

Function name	Signature	Parameter type
xs:hexBinary	xs:hexBinary(*&exp*) xs:hexBinary(*&exp*, *&ccsid*) xs:hexBinary(*&exp*, *&ccsid*, *&encoding*)	*$exp* is a reference to a source element of one of the following types: • xs:base64Binary • xs:int • xs:string *&ccsid* is one of the following values: • $source/Properties/CodedCharSetId • $source/MQMD/CodedCharSetId • An integer. For example, 1208 for UTF-8 *&encoding* is one of the following values: • $source/Properties/Encoding • $source/MQMD/Encoding • $mq:MQENC_WINDOWS (546) • $mq:MQENC_UNIX (273) • $mq:MQENC_390 (785) • Other $mq:MQENC_* constants and their BIT OR
xs:int	xs:int(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:base64Binary • xs:decimal • xs:duration • xs:float • xs:hexBinary • xs:string
xs:integer	xs:integer(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:base64Binary • xs:decimal • xs:duration • xs:int • xs:string

Function name	Signature	Parameter type
xs:long	xs:long(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:base64Binary • xs:decimal • xs:duration • xs:float • xs:hexBinary • xs:string

Function name	Signature	Parameter type
xs:QName	xs:QName(*&exp*) xs:QName(*&exp*, *&ccsid*) xs:QName(*&exp*, *&ccsid*, *&encoding*)	*$exp* is a reference to a source element of one of the following types: • xs:anyURI • xs:base64Binary • xs:boolean • xs:date • xs:dateTime • xs:dayTimeDuration • xs:decimal • xs:double • xs:duration • xs:float • xs:gDay • xs:gMonth • xs:gMonthDay • xs:gYear • xs:gYearMonth • xs:hexBinary • xs:int • xs:integer • xs:long • xs:string • xs:time • xs:yearMonthDuration *&ccsid* is one of the following values: • $source/Properties/CodedCharSetId • $source/MQMD/CodedCharSetId • An integer. For example, 1208 for UTF-8 *&encoding* is one of the following values: • $source/Properties/Encoding • $source/MQMD/Encoding • $mq:MQENC_WINDOWS (546) • $mq:MQENC_UNIX (273) • $mq:MQENC_390 (785) • Other $mq:MQENC_* constants and their BIT OR

Function name	Signature	Parameter type
xs:string	xs:string(*&exp*) xs:string(*&exp*, *&ccsid*) xs:string(*&exp*, *&ccsid*, *&encoding*)	*$exp* is a reference to a source element of one of the following types: • xs:anyURI • xs:base64Binary • xs:boolean • xs:date • xs:dateTime • xs:dayTimeDuration • xs:decimal • xs:double • xs:duration • xs:float • xs:gDay • xs:gMonth • xs:gMonthDay • xs:gYear • xs:gYearMonth • xs:hexBinary • xs:int • xs:integer • xs:long • xs:QName • xs:time • xs:yearMonthDuration *&ccsid* is one of the following values: • $source/Properties/CodedCharSetId • $source/MQMD/CodedCharSetId • An integer. For example, 1208 for UTF-8 *&encoding* is one of the following values: • $source/Properties/Encoding • $source/MQMD/Encoding • $mq:MQENC_WINDOWS (546) • $mq:MQENC_UNIX (273) • $mq:MQENC_390 (785) • Other $mq:MQENC_* constants and their BIT OR
xs:time	xs:time(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:dateTime • xs:string

Function name	Signature	Parameter type
xs:yearMonthDuration	xs:yearMonthDuration(*&exp*)	*$exp* is a reference to a source element of type xs:string.

Headers and Mapping node

This topic lists the headers that can be manipulated by the Mapping node.

You can map these headers:

- MQ Headers
  - MQMD
  - MQCFH header with root element MQPCF
  - MQCIH
  - MQDLH
  - MQIIH
  - MQMDE
  - MQRFH
  - MQRFH header with MQRFH2 or MQRFH2C parser
  - MQRMH
  - MQSAPH
  - MQWIH
  - SMQ_BMH
- Email Headers
  - EmailOutputHeader
- HTTP Headers
  - HTTPInputHeader
  - HTTPReplyHeader
  - HTTPRequestHeader
  - HTTPResponseHeader
- JMSTransport

Migrating message mappings from Version 5.0

Use the mqsimigratemfmaps command to migrate message mappings to the Version 6.1 format.

The mqsimigratemfmaps command creates Version 6.1 mapping files (`.msgmap`) from your Version 5.0 mapping files (`.mfmap`).

When you migrate message mappings from Version 5.0, read the restrictions that apply.

The following table lists the mapping functions that are supported in Version 5.0 but not supported in Version 6.1, and shows the error messages that you might see. Mappings that contain these Version 5.0 functions cannot be migrated to Version 6.1; you must re-create and redeploy these mappings using another node, such as a JavaCompute node. Alternatively, migrate as much of the mapping as possible using the migration command, view the error report to see details of the functions that could not be migrated, and create a new node that can execute those functions that were not migrated.

Supported in Version 5.0	Migration utility error message
Expressions that involve multiple instances of a repeating source element, for example: `src_msg.e[1] + src_msg.e[2] -> tgt_msg.e`	`Error:102: Unexpected index '2' encountered for target mappable 'e'. The expected index is '1'. Migration currently provides no support for expressions involving more than one instance of the same repeating-element.`
ESQL field references that contain the asterisk wildcard character "*". For example: `src_msg.e.* or src_msg.e.*[]`	`Error:130: ESQL field-reference 'src_msg.e.*' cannot be migrated. Migration currently provides no support for field-references containing '*'.`
Dynamic ESQL field references. For example: `src_msg.e.{'a' || 'b'}`	`Error:131: ESQL field-reference 'src_msg.e.{'a' || 'b'}' cannot be migrated. Migration currently provides no support for dynamic field-references.`
ESQL expressions that contain a reference to the temporary index-variable "#I". For example: `src_msg_e || "#I" -> tgt_msg.e`	`Error:128: ESQL expressions containing the variable '#I' anywhere other than the index of a repeating-element cannot be handled by the migration.`
Expressions within an index of a repeating element. For example: `src_msg.e[src_msg.a] or src_msg.e["#I" +5] or src_msg.e[< 3]`	`Error:116: ESQL field-reference 'src_msg.e[< 3]' cannot be migrated. Migration currently provides no support for indexes other than the variable '#I' and plain integer indexes.`
Aggregation functions MIN, MAX, and COUNT, used with the ESQL SELECT expression. For example: `SELECT MAX("#T".FIRSTNAME) FROM Database.CUSTOMER AS "#T" WHERE "#T".CUSTOMERID = 7`	`Error:135: The ESQL expression 'SELECT MAX("#T".FIRSTNAME) FROM Database.CUSTOMER AS "#T" WHERE "#T".CUSTOMERID = 7' could not be migrated. The expression contains syntax which has no direct equivalent in the new map-script language.`
ESQL IN operator. For example: `src_msg.e IN (1, 2, 3)`	`Error:135: The ESQL expression 'SELECT MAX("#T".FIRSTNAME) FROM Database.CUSTOMER AS "#T" WHERE "#T".CUSTOMERID = 7' could not be migrated.`

Restrictions on migrating message mappings

Learn how to migrate message maps from Version 5.0.

The programming model for message maps is different between Version 5.0 (where the file format is `.mfmap`) and Version 6.1 (where the format is `.msgmap`). Version 5.0 message maps have a procedural programming model, which is essentially an alternative ESQL, where you describe all the steps that are required to perform a transformation. Version 6.1 uses a declarative programming model, where you describe the result of the transformation, and the tools determine how to achieve that result.

Most migration failures result from message maps that contain too much information about the steps that perform the transformation, and not enough information about the required result. For these message maps, migration is enabled by changing the `.mfmap` file so that specific "how to" sections are separated into an ESQL function or procedure that can be called by the message map. The

.mfmap file calls the ESQL function instead of containing it as an expression. The mqsimigratemfmaps command then migrates the .mfmap file, but calls the ESQL function instead of logging a migration error.

A limitation is that ESQL (the run time for .mfmap and .msgmap files) cannot define functions that return complex element (or REFERENCE) values. The following procedure explains how to work around this complex element target limitation; in many cases, you must rewrite the map as an ESQL function. For more examples and information about calling ESQL from maps, see the following sample:

- Message Map

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

1. Determine whether you can define an ESQL function for the .mfmap file.
   a. When the target value is a complex element, or in ESQL terms a REFERENCE, the individual mapping must be rewritten in the .msgmap file. Delete the mapping from the .mfmap file, and proceed to Step 4.
   b. Use a function for all other cases: CHAR string, numbers, date, and time. Proceed to Step 2.
2. Determine the source parameters and returns type for your function.
   a. For each source path in the mapping, there must be one parameter in the function or procedure. For a function, all parameters are unchangeable. The type of the parameter must match the source data type.
   b. The function return type is the ESQL data type identified previously.
3. Update the .mfmap file to enable migration. Change the .mfmap file to invoke the function in the mapping, passing the source parameters to the function in the order in which they were listed in step 2a.
4. Re-run the mqsimigratemfmaps command to migrate the modified .mfmap file.
5. Repeat Steps 1 to 4 until no errors are reported in the migration log.
6. Start the Version 6.1 Message Broker Toolkit and open the migrated .msgmap file.
   a. For ESQL that is migrated as functions, there should be no errors.
   b. For complex element targets, rewrite the mapping by using the Version 6.1 features.

The following examples illustrate migration of .mfmap files to .msgmap files.

- To migrate a multiple reference to a repeating source expression:

  ```
  src_msg.e[1] + src_msg.e[2]
  ```

  compute the result in an ESQL function such as:

  ```
  CREATE FUNCTION addOneAndTwo(IN src_msg)
  BEGIN
   RETURN src_msg.e[1] + src_msg.e[2];
  END;
  ```

  In the .msgmap file, call the ESQL function addOneAndTwo by using the parent element **src_msg** as a parameter.

- An expression that does not use element names:

  ```
  src_msg.*
  ```

  or

  ```
  src_msg.*[]
  ```

can be processed using a function that takes the parent of the repeating field:

```
CREATE FUNCTION processAny(IN src_msg)
BEGIN
 DECLARE nodeRef REFERENCE TO src_msg.e.*;
 DECLARE result <dataType> <initialValue>;
 WHILE LASTMOVE nodeRef DO
  --expression goes here
  SET result = result;
 END WHILE;
 RETURN RESULT;
END;
```

In the `.msgmap` file, call the ESQL function processAny by using the parent element **src_msg** as a parameter.

- Expressions that dynamically compute element names:

```
src_msg.{'a' || 'b'}
```

can be processed by ESQL functions that process the parent of the repeating field:

```
CREATE FUNCTION processDynamicName(IN src_msg)
BEGIN
 RETURN src_msg.{'a' || 'b'};
END;
```

In the `.msgmap` file, call the ESQL function processDynamicName by using the parent element **src_msg** as a parameter.

- Expressions that use the select MIN, MAX, and COUNT functions:

```
SELECT MAX("#T".FIRSTNAME)
 FROM Database.CUSTOMER AS "#T"
 WHERE "#T".CUSTOMERID = custId
```

can be processed by ESQL functions that process the parent of the repeating field:

```
CREATE FUNCTION processMAX(IN custId)
BEGIN
 RETURN
 SELECT MAX("#T".FIRSTNAME)
  FROM Database.CUSTOMER AS "#T"
  WHERE "#T".CUSTOMERID = custId
END;
```

In the `.msgmap` file, call the ESQL function processMAX by using the element **custId** as a parameter.

- `.mfmap` files that use mfmap index variables in expressions:

```
e || "#I"
```

must be rewritten entirely in ESQL. By definition, there must be a complex repeating parent element, and this is not supported by ESQL functions.

- Expressions that use source expressions to compute values:

```
src_msg.e[src_msg.a]
```

must be rewritten by using `if` rows, msgmap:occurrence() functions, and ESQL functions:

```
for src_msg.e
 if
  condition msgmap:occurrence(src_msg/e) = src_msg/a
```

- For expressions that use index expressions to compute values:

```
src_msg.e["#I" +5]
src_msg.e[< 3]
```

the entire `.msgmap` file must be rewritten in ESQL, because the `.msgmap` files do not support indexed access to repeating fields.

- `.mfmap` files that use ROW expressions to compute values:

```
src_msg.e IN (1, 2, 3)
```

must be rewritten in ESQL, because `.msgmap` files do not support ESQL ROW expressions.

Restrictions on migrating maps that call ESQL

If there is a mismatch between the case that has been in the ESQL call in the message map, and the name of the routine defined in the ESQL file, an error is produced during migration of the message map. To prevent an error occurring during migration, ensure that the ESQL call in the message map uses the same case as the ESQL defined in the routines in the ESQL file. Alternatively, you can manually edit the message map after migration to call the ESQL routine with matching case.

Restrictions on migrating submaps

In Version 5.0 message maps, any complex element type can be the root of a submap. However, in Version 6.1, only a global element or a global attribute can be the root of a submap. When a Version 5.0 message map with a call to a submap with a non-global element as the map root is migrated, the submap is not migrated as a stand-alone submap. Instead, the call to the submap in the main message map is replaced by the migrated content of the submap. Alternatively, if the submap has a global element as the map root, the submap is migrated to a stand-alone Version 6.1 submap instead.

For Version 6.1, you must define reusable schema structures as global elements and types. If you have Version 5.0 submaps that use local elements, you must change the schema to add definitions of global elements for the local elements, and then use the new schema after migration. If the new global elements have the same name and type as the local elements, the Version 5.0 submaps do not have to be changed.

You must qualify a local element in a Version 5.0 submap with a namespace to ensure its successful migration to Version 6.1, because the global element that replaces it after migration must be qualified by the namespace. If your submap contains local elements, you must re-create the submap and re-create the call to the submap from the main message map.

The following table shows differences between the features that are supported in a submap for Version 5.0 and Version 6.1.

Version	Supported feature
Version 5.0	global elements and global attributes as map source
	global elements and global attributes as map target
	local elements and local attributes as map source
	local elements and local attributes as map target
Version 6.1	global elements, global attributes, and global types as map source
	global elements and global attributes as map target

XML constructs

A self-defining XML message carries the information about its content and structure within the message in the form of a document that adheres to the XML specification. Its definition is not held anywhere else.

When the broker receives an XML message, it interprets the message using the generic XML parser, and created an internal message tree structure according to the XML definitions contained within that message.

A self-defining message is also known as a generic XML message. It does not have a recorded format.

The information provided with WebSphere Message Broker does not provide a full definition or description of XML terminology, concepts, and message constructs: it is a summary that highlights aspects that are important when you use XML messages with brokers and message flows.

For further information about XML, see the developerWorks Web site.

Example XML message

The name elements used in this description (for example, XmlDecl) are provided by WebSphere Message Broker, and are referred to as field type constants. They are available for symbolic use within the ESQL that defines the processing of message content performed by the nodes, such as a Filter node, within a message flow. They are not part of the XML specification.

A simple XML message might take the form:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>.........</s1>
```

The corresponding syntax element tree (top level elements only) is shown in the following diagram:

The WhiteSpace elements within the tree are there because of the line breaks in the original XML document, and have no business meaning. White space is used in XML for readability; if you process XML messages that contain line breaks (as shown above), blanks lines, or spaces between tags, these all appear as elements in the message tree.

WhiteSpace within an XML element (between start and end tags) has business meaning and is represented using the Content syntax element. See "XML WhiteSpace and DocTypeWhiteSpace" on page 1579 for more information.

The field type constants for XML name elements (for example, Element and XmlDecl) equate to a constant value of the form `0x01000000`. You can see these constants in the output created by the Trace node when a message, or a portion of the message, is traced.

The XML declaration

The beginning of an XML message can contain an XML declaration.

The following is an example of a declaration:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>.........</s1>
```

The XML declaration includes the following field type constants:

- "XML encoding"
- "XML standalone"
- "XML version" on page 1569
- "XMLDecl" on page 1569

"XML declaration example" on page 1569 includes another example of an XML declaration and the tree structure it forms.

XML encoding

The encoding element is a value element and is always a child of the XmlDecl element.

The value of the encoding element is a string that corresponds to the value of the encoding string in the declaration. In the following example, the encoding element has a value of `UTF-8`.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>.........</s1>
```

You cannot specify WebSphere MQ encodings in this element.

XML standalone

The XML standalone element defines the existence of an externally-defined DTD.

It is a value element and stores the data corresponding to the value of the standalone string in the declaration. It is always a child of the XmlDecl element. Valid values for the standalone element are `yes` and `no`. The following is an example of this:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>.........</s1>
```

A value of no indicates that this XML document is not standalone and depends on an externally-defined DTD. A value of yes indicates that the XML document is self-contained. However, the current release of WebSphere Message Broker does not resolve externally-defined DTDs, so the setting of standalone is irrelevant and is ignored.

XML version

The XML version element is a value element and stores the data corresponding to the version string in the declaration.

It is always a child of the XmlDecl element. In the following example, the version element contains the string value 1.0:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>.........</s1>
```

XMLDecl

XMLDecl is a name element that corresponds to the XML declaration itself.

The XmlDecl element is a child of the XML parser and is written first to a bit stream. Although the XMLDecl element is a named element, its name has no relevance. The following shows an example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>.........</s1>
```

XML declaration example

The following example shows an XML declaration in an XML document.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

The following figure shows the tree structure that is created from the declaration:



The XML message body

Every XML message must have a body. The body comprises a top-level XML element that contains all the message data.

The body contains complex XML markup, which translates to many syntax element types in the parsed tree. Each syntax element type is introduced here, with a series of example XML fragments.

The following common element types are discussed:

- "XML element" on page 1572
- "XML attribute" on page 1570
- "XML content" on page 1572

"XML message body example: elements, attributes, and content" on page 1573 provides an example of an XML message body and the tree structure that is created from it using the syntax elements types listed above.

More complex XML messages might use some of the following syntax element types:
- "XML CDataSection" on page 1571
- "XML EntityReferenceStart and EntityReferenceEnd" on page 1572
- "XML comment" on page 1571
- "XML ProcessingInstruction" on page 1573
- "XML AsisElementContent"
- "XML BitStream" on page 1571

XML AsisElementContent

AsisElementContent is a special value syntax element that is used to precisely control generated XML.

The AsisElementContent syntax element is a special value element. The element is used to precisely control the XML generated in an output message without the safeguards of the Element, Attribute, and Content syntax elements. If you use AsisElementContent, you must ensure that the output message is well-formed XML.

You might choose to use this syntax element if, for example, you want to suppress the usual behavior in which occurrences of ampersand (&), less than (<), greater than (>), quotation mark ("), and apostrophe (') are replaced by the predefined XML entities `&`, `<`, `>`, `&quot;`, and `&apos;`.

The following example illustrates the use of AsisElementContent. The statement:

```
Set OutputRoot.XMLNS.(XML.Element)Message.(XML.Content) = '<rawMarkup>';
```

generates the following XML in an output message:

```
<Message>&lt;rawMarkup&gt;</Message>
```

However, the following statement:

```
Set OutputRoot.XMLNS.(XML.Element)Message.(XML.AsisElementContent) = '<rawMarkup>';
```

generates the following output message:

```
<Message><rawMarkup></Message>
```

These examples show that the value of an AsisElementContent syntax element is not modified before it is written to the output message.

XML attribute

This syntax element is the default name-value element supported by the XML parser. Use it to represent the attributes that are associated with its parent element. The name and value of the syntax element correspond to the name and value of the attribute that is being represented. Attribute elements have no children, and must always be children of an element.

When attributes are written to a message, occurrences of ampersand (&), less than (<), greater than (>), double quotation mark ("), and apostrophe (') within the attribute value are replaced by the predefined XML entities `&`, `<`, `>`, `&quot;`, and `&apos;`.

The attr element is also supported for compatibility with earlier versions of the product.

XML BitStream

BitStream is a specialized name-value element designed to aid the processing of very large messages.

The XML BitStream syntax element is a name-value element. When writing an XML message, the value of the BitStream element is written directly into the message, and the name is not important. The BitStream element might be the only element in the message tree.

The value of the element must be of type BLOB; any other data type generates an error while writing the element. Ensure that the content of the element is appropriate for use in the output message.

Use of the BitStream element is similar to the AsisElementContent element, except that the AsisElementContent type converts its value into a string, whereas the BitStream element uses its BLOB value directly. This is a specialized element designed to aid processing of very large messages.

The following ESQL excerpts demonstrate a typical use for this element. First, declare the element:

```
DECLARE StatementBitStream BLOB
```

Initialize the contents of StatementBitStream from an appropriate source, such as an input message. If the source field is not of type BLOB, use the CAST statement to convert the contents to BLOB. Then create the new field in the output message, for example:

```
CREATE LASTCHILD OF resultCursor
 Type XML.BitStream
 NAME 'StatementBitStream'
 VALUE StatementBitstream;
```

XML CDataSection

CData sections in the XML message are represented by the CDataSection value element. The content of the CDataSection element is the value of the CDataSection element without the <![CDATA[that marks its beginning and the]]> that marks its end.

For example, the following Cdata section:

```
<![CDATA[<greeting>Hello, world!</greeting>]]>
```

is represented by a CDataSection element with a string value of:

```
"<greeting>Hello, world!</greeting>"
```

Unlike Content, occurrences of <,>, &, ", and ' are not translated to their escape sequences when the CDataSection is written out to a serialized message (bit stream).

XML comment

An XML comment encountered outside the document type declaration is represented by the Comment value syntax element. The value of the element is the comment text from the XML message.

If the value of the element contains the character sequence `-->`, the sequence is replaced with the text `-->`. This ensures that the contents of the comment cannot prematurely terminate the comment. Occurrences of <, >, &, ", and ' are not translated to their escape sequences.

The following are examples of the XML comment in an XML document and in tree structure form:

```
<example><!-- This is a comment --></example>
```



XML content

This syntax element is the default value element supported by the XML parser. Use content to represent character data (including white space characters) that is part of the element content. There might be many content elements as children of a single element, in which case they are separated by other syntax element types such as nested elements or attributes.

When content is written to a message, occurrences of ampersand (&), less than (<), greater than (>), double quotation mark ("), and apostrophe (') are replaced by the predefined XML entities `&`, `<`, `>`, `&quot;`, and `&apos;`.

The pcdata element is also supported for compatibility with earlier versions of the product.

XML element

This syntax element is the default name element supported by the XML parser, and is one of the most common elements. The name of the syntax element corresponds to the name of the XML element in the message. This element can have many children, including attributes, elements, and content.

The tag element is also supported for backward compatibility.

XML EntityReferenceStart and EntityReferenceEnd

When an entity reference is encountered in the XML message, both the expanded form and the original entity name are stored in the syntax element tree. The name of the entity is stored as the value of the EntityReferenceStart and EntityReferenceEnd syntax elements, and any syntax elements between contain the entity expansion.

The following examples show the XML entity references in an XML document and in tree structure form:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE example [ <!ENTITY entityName "eValue"> ]>
<example>Test: &entityName;</example>
```



The XML declaration and the document type declaration are not shown here. Refer to "The XML declaration" on page 1568 and "XML document type declaration" on page 1574 for details of those sections of the syntax element tree.

XML message body example: elements, attributes, and content

The following are examples of an XML message body in an XML document and in tree structure form. The XML document contains elements, attributes, and content, and these items are shown in the tree structure.

```
<Person age="32" height="172cm">
<Name>Cormac Keogh</Name>
</Person>
```



XML ProcessingInstruction

ProcessingInstruction is a syntax element used with XML processing instructions.

An XML processing instruction, encountered outside the document type declaration, is represented by the ProcessingInstruction syntax element. This is a name-value element; the name of the syntax element is the processing instruction target name, and the value of the syntax element is the character data of the processing instruction. The value of the syntax element must not be empty. The name cannot be XML in either uppercase or lowercase.

If the value of the element contains the character sequence `?>`, the sequence is replaced with the text `?>`. This ensures that the content of the processing instruction cannot prematurely end the processing instruction. Occurrences of <,>, &, ", and ' are not translated to their escape sequences.

Examples of the XML ProcessingInstruction in an XML document and in tree structure form are shown in the following:

```
<example><?target This is a PI.?></example>
```



XML document type declaration

The document type declaration (DTD) of an XML message is represented by a syntax element of type DocTypeDecl and its children and descendants. These comprise the DOCTYPE construct.

Only internal (inline) DTD subsets are represented in the syntax element tree. An inline DTD is a DTD that is declared within the XML document itself. It can be a complete DTD definition, or can extend the definition in an external DTD.

External DTD subsets (identified by the SystemID or PublicId elements described later in this section) can be referenced in the message, but those referenced are not resolved by the broker.

Field type constants are defined by WebSphere Message Broker:
- DocTypeDecl
- NotationDecl
- Entities
- ElementDef
- AttributeList
- AttributeDef
- DocTypePI
- WhiteSpace and DocTypeWhiteSpace
- DocTypeComment

DTD example is an example of an XML DTD.

XML DocTypeDecl

DocTypeDecl is a named element and is a child of the XML parser. DocTypeDecl is written to the bit stream before the element that represents the body of the document during serialization. The following attributes can be specified within this element:
- IntSubset
- PublicId
- SystemId

The following example is included in DTD example:

```
<!DOCTYPE test PUBLIC "//this/is/a/URI/test" "test.dtd" [
...
...
]>
```

XML IntSubset:

IntSubset is a named element that groups all those elements that represent the DTD constructs contained in the internal subset of the message. Although the IntSubset element is a named element, its name is not relevant.

XML PublicId:

PublicId is an element that represents a public identifier in an XML message. It can be part of a DocTypeDecl, NotationDecl, or UnparsedEntityDecl element. The value of the PublicId element is typically a URL. A public identifier of the form `PUBLIC "//this/is/a/URI/test"` has a string value of `//this/is/a/URI/test`.

XML SystemId:

SystemId is a value element that represents a system identifier in an XML message. It can be part of a DocTypeDecl, NotationDecl, or UnparsedEntityDecl element. The value of the SystemId is a URI, and is typically a URL or the name of a file on the current system. A system identifier of the form `SYSTEM "Note.dtd"` has a string value of `Note.dtd`.

XML NotationDecl

The NotationDecl element represents a notation declaration in an XML message. NotationDecl is a name element whose name corresponds to the name given with the notation declaration. It must have a SystemId as a child and it can optionally have a child element of type PublicId. For example:

```
<!NOTATION gif SYSTEM "image.gif">
```

The name of the NotationDecl is `gif`.

XML entities

Entities in the DTD are represented by one of six named element types described in the following topics:

- EntityDecl
- EntityDeclValue
- ExternalParameterEntityDecl
- ExternalEntityDecl
- ParameterEntityDecl
- UnparsedEntityDecl

XML EntityDecl:

The EntityDecl element represents a general entity and is declared in the internal subset of the DTD. It is a named element and has a single child element, which is of type EntityDeclValue.

An entity declaration of the form:

```
<!ENTITY bookTitle "User Guide">
```

has an EntityDecl element of name bookTitle and a child element of type EntityDeclValue with a string value of `User Guide`.

XML EntityDeclValue:

The EntityDeclValue element represents the value of an EntityDecl or ParameterEntityDecl defined internally in the DOCTYPE construct. It is always a child of an element of one of those types, and is a value element. For the following entity:

```
<!ENTITY  bookTitle "User Guide">
```

the EntityDeclValue element has the string value `User Guide`.

XML ExternalParameterEntityDecl:

The ExternalParameterEntityDecl element represents a parameter entity definition where the entity definition is contained externally to the current message. It is a named element and has a child of type SystemId. It can also have a child of type PublicId. The name of the entity does not include the percent sign %. In XML an external parameter entity declaration takes the form:

```
<!ENTITY % bookDef SYSTEM "BOOKDEF.DTD">
```

This represents an ExternalParameterEntityDecl element of name bookDef with a single child of type SystemId with a string value of `BOOKDEF.DTD`.

XML ExternalEntityDecl:

The ExternalEntityDecl element represents a general entity where the entity definition is contained externally to the current message. It is a named element and has a child of type SystemId. It can also have a child of type PublicId.

An external entity declaration of the form:

```
<!ENTITY  bookAppendix SYSTEM "appendix.txt">
```

has an EntityDecl element of name bookAppendix and a child element of type SystemId with a string value of `appendix.txt`.

XML ParameterEntityDecl:

The ParameterEntityDecl represents a parameter entity definition in the internal subset of the DTD. It is a named element and has a single child element that is of type EntityDeclValue. For parameter entities, the name of the entity does not include the percent sign %. In XML a parameter entity declaration takes the form:

```
<!ENTITY % inline "#PCDATA | emphasis | link">
```

XML UnparsedEntityDecl:

An unparsed entity is an external entity whose external reference is not parsed by an XML processor. This means that you can include data in an XML document that is not well-formed XML, such as a graphic file. The UnparsedEntityDecl is named element and a child of type SystemId that identifies the URI for the entity (a URL or a local file location). UnparsedEntityDecl can optionally have a child of type PublicId.

UnparsedEntityDecl can also have a child of type NotationReference, a value element that represents a reference to a notation declaration elsewhere in the XML document. It defines the type of data of the unparsed entity.

An unparsed entity declaration takes the form:

```
<!ENTITY pic SYSTEM "scheme.gif" NDATA gif>
```

In this example, the SystemId has a string value of `scheme.gif`. The value of NotationReference is `gif`. It refers to a NOTATION defined within the XML document:

```
<!NOTATION gif SYSTEM "image/gif">
```

The next entity is included in the DTD example:

```
<!ENTITY unpsd PUBLIC "//this/is/a/URI/me.gif" "me.gif" NDATA TeX>
```

This shows the optional PublicId element, which has the string value of `//this/is/a/URI/me.gif`.

XML ElementDef

The ElementDef element represents the `<!ELEMENT` construct in a DTD. It is a child of the DOCTYPE element. The name of the element that is defined corresponds to the name of the syntax element. The value corresponds to the element definition.

This example is included in the DTD example:

```
<!ELEMENT subel2 (#PCDATA)>
```

The name of the element is `subel2` and the value is `(#PCDATA)`.

XML AttributeList

The AttributeList name element represents the `<!ATTLIST` construct in a DTD. The name of the AttributeList element corresponds to the name of the element for which the list of attributes is being defined. Its content represents one or more AttributeDef elements.

This example is included in the DTD example:

```
<!ATTLIST el5  el5satt CDATA #IMPLIED>
```

This example shows an AttributeList that defines one AttributeDef, and its content is explained in AttributeDef.

XML AttributeDef

The AttributeDef name element describes the definition of an attribute within an `<!ATTLIST` construct. It is always a child of the AttributeList element. The name of the syntax element is the name of the attribute being defined. It can have three children:

- AttributeDefValue
- AttributeDefType
- AttributeDefDefaultType

This example is included in the DTD example:

```
<!ATTLIST el5  el5satt CDATA #IMPLIED>
```

The name of the AttributeDef is `el5satt` and it is a child of AttributeList `el5`. The name of the AttributeDefType is `CDATA`, and the value of the AttributeDefDefaultType is `IMPLIED`.

XML AttributeDefValue:

For attributes of type CDATA (see "XML AttributeDefType"), or defined by an enumerated list, the AttributeDefValue gives the default value of the attribute.

For an example of AtrtibuteDefValue, see DTD example.

XML AttributeDefType:

The AttributeDefType syntax element is a name-value element whose name corresponds to the attribute type found in the attribute definition. Possible values for the name are:
- CDATA
- ID
- IDREF
- IDREFS
- ENTITY
- ENTITIES
- NMTOKEN
- NMTOKENS
- NOTATION

If there is an enumeration present for the attribute definition, the entire enumeration string is held as a string in the value member of the name-value syntax element. In this case, the name member of the name-value syntax element is empty. The value string starts with an open parenthesis (and ends with a close parenthesis). Each entry in the enumeration string is separated by a vertical bar | character. If the Attribute value is not defined by an enumerated list, the value member of the syntax element is empty.

An example is included in AttributeDef.

XML AttributeDefDefaultType:

The AttributeDefDefaultType syntax element is a value element that represents the attribute default as defined under the attribute definition. The value can be one of the following strings:
- #REQUIRED
- #IMPLIED
- #FIXED

An example is included in AttributeDef.

XML DocTypeComment

Comments in the XML DTD are represented by the DocTypeComment element. It is a value element for which the value string contains the comment text. This element follows the same processing rules as the Comment element. See "XML comment" on page 1571.

XML DocTypePI

The DocTypePI element represents a processing instruction found within the DTD. The ProcessingInstruction element represents a processing instruction found in the XML message body.

This element is a name-value element. The name of the element is used to store the processing instruction target name, and the value contains the character data of the

processing instruction. The value of the element can be empty. The name cannot be the string XML in either uppercase or lowercase form. This element follows the same processing rules as the ProcessingInstruction element. See "XML ProcessingInstruction" on page 1573.

XML WhiteSpace and DocTypeWhiteSpace

The WhiteSpace element represents any white space characters outside the message body and DTD that are not represented by any other element. For example, white space within the body of the message (within elements) is reported as element content using the Content element type, but white space characters between the XML declaration and the beginning of the message body are represented by the WhiteSpace element.

```
<?xml version="1.0"?>       <BODY>....</BODY>
```

The characters between `"1.0"?>` and `<BODY>` are represented by the WhiteSpace element.

White space is used in XML for readability and has no business meaning. Input XML messages can include line breaks, blanks lines, and spaces between tags (all shown in the following example). If you process XML messages that contain any of these spaces, they are represented as elements in the message tree. Therefore they appear when you view the message in the debugger, and in any trace output.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>.........
<s2>abc</s2>   <s2>def</s2>

<s3>123</s3>
</s1>
```

If you do not want white space elements in your message trees, you must present the input message as a single line, or use the XMLNSC compact parser in its default mode

The DocTypeWhiteSpace element represents white space that is found inside the DTD that is not represented by any other element. White space characters found within a DocType between two definitions are represented by the DocTypeWhiteSpace element.

```
<!ENTITY % bookDef SYSTEM "BOOKDEF.DTD">  <!ENTITY bookTitle "User Guide">
```

The characters between `DTD">` and `<!ENTITY` are represented by the DocTypeWhiteSpace element.

XML DTD example

This example shows an XML DTD in an XML document and the tree structure form of that document:

```
<!DOCTYPE test PUBLIC "//this/is/a/URI/test" "test.dtd" [
<!NOTATION TeX  PUBLIC "//this/is/a/URI/TexID" "//TexID">
<!ENTITY ent1 "this is an entity">
<!ENTITY % ent2 "#PCDATA | subel2">
<!ENTITY % extent1 PUBLIC "//this/is/a/URI/extent1" "more.txt">
<!ENTITY extent2 PUBLIC "//this/is/a/URI/extent2" "more.txt">
<!ENTITY unpsd PUBLIC "//this/is/a/URI/me.gif" "me.gif" NDATA TeX>
<?test Do this?>
<!--this is a comment-->
<!ELEMENT subel2 (#PCDATA)>
```

```
<!ELEMENT subel1 (subel2 | el4)+>
<!ELEMENT el1 (#PCDATA)>
<!ELEMENT el2 (#PCDATA | subel2)*>
<!ELEMENT el3 (#PCDATA | subel2)*>
<!ELEMENT el4 (#PCDATA)>
<!ELEMENT el5 (#PCDATA | subel1)*>
<!ELEMENT el6 (#PCDATA)>
<!ATTLIST subel1
 size (big | small) "big"
 shape (round | square) #REQUIRED>
<!ATTLIST el5
 el5satt CDATA #IMPLIED>
]>
```

When a message is parsed by the generic XML parser, the relevant part of the message tree looks like this (assuming that there are no carriage returns or white space between tags):



The IntSubset structure contains the following structures at the next level of nesting: the tree structure for each of these is shown in the following tree structures.






ParameterEntityDecl
- name="ent2"
EntityDeclValue
- value="#PCDATA | subel2"
ExternalParameterEntityDecl
- name="extent1"
SystemId
- value="more.txt"
PublicId
- value="//this/is/a/URI/extent2"
ExternalEntityDecl
- name="extent2"
SystemId
- value="more.txt"
PublicId
- value="//this/is/a/URI/extent2"
UnparsedEntityDecl
- name="unpsd"
SystemId
- value="me.gif"
PublicId
- value="//this/is/a/URI/me.gif"
NotationReference
- value="TeX"
DocTypeWhiteSpace
- value="
"
DocTypePI
- name="test"
- value="Do this"
DocTypeComment
- value="this is a comment"


ElementDef
- name="subel2"
- value="(#PCDATA)"
ElementDef
- name="subel1"
- value="Subel2 | el4"
ElementDef
- name="el1"
- value="(#PCDATA)"
ElementDef
- name="el2"
- value="(#PCDATA | Subel2)*"
ElementDef
- name="el3"
- value="(#PCDATA | Subel2)*"
ElementDef
- name="el4"
- value="(#PCDATA)"
ElementDef
- name="el5"
- value="(#PCDATA | Subel1)*"
ElementDef
- name="el6"
- value="(#PCDATA)"



Data sources on z/OS

The Data Source name in the Compute and Database nodes identifies the location of the table referred to in the respective node's ESQL. Data sources on z/OS correspond to DB2 subsystems rather than DB2 databases. The DB2 owning region for a particular database table is identified using a combination of the DSNAOINI file and DB2 subsystem configuration.

The MVSDEFAULTSSID parameter in the DSNAOINI file identifies the local DB2 subsystem to which the broker is connected. This subsystem is used to locate the data source which is either a local or remote DB2. The mapping between a particular data source and DB2 subsystem is shown in the DSNTIPR installation panel of the default DB2 subsystem and SYSIBM.LOCATIONS table.

When you access remote DB2 subsystems, ensure that the DBRMs for ODBC are bound at the remote subsystem. For more information, refer to the 'Programming for ODBC' topics in the DB2 Information Management Software Information Center for z/OS Solutions .

If you need to access databases that are not on DB2 on z/OS, you can use DB2's Distributed Data Facility (DDF) and Distributed Relational Architecture (DRDA) to incorporate a remote unit of work within a message flow.

Message mappings

Edit and configure message maps using the Message Mapping editor.

This section contains topics that provide reference information about message mapping:

- "Message Mapping editor"
  - Source pane
  - Target pane
  - Edit pane
  - Spreadsheet pane
- "Mapping node" on page 1597
  - Syntax
  - Functions
  - Casts
- "Migrating message mappings from Version 5.0" on page 1617
  - Migration restrictions

Message Mapping editor

You configure a message mapping using the Message Mapping editor, which you use to set values for:

- the message destination
- message headers
- message content

Here is an example of the Message Mapping editor. There are separate panes for working with sources, targets and expressions, as well as a spreadsheet view.



1. **Source pane:** displays a source message or database table
2. **Target pane:** displays a target message
3. **Edit pane:** displays the expression to be used to derive the target element value
4. **Spreadsheet pane:** displays a summary of the mappings in spreadsheet columns (each target field and its value)

Use the Message Mapping editor to perform various mapping tasks.

Wizards and dialog boxes are provided for tasks such as adding mappable elements, working with ESQL, and working with submaps. Mappings that are created with the Message Mapping editor are automatically validated and compiled, ready for adding to a broker archive (BAR) file, and subsequent deployment to WebSphere Message Broker.

Message Mapping editor Source pane

Details of the elements present in the Source pane of the Message Mapping Editor.

The following example shows the “Message Mapping editor” on page 1585. The pane that is labelled as 1 in the example is the Source pane:



The following list describes the elements that are present in the Source pane:

- A source message is identified by $source.
- A source database is identified by $db:select.
- A database stored procedure is identified by $db:proc.
- A database user-defined function is identified by $db:func.
- A mapped entry is indicated by a blue triangle alongside the element. In this example, Customer_ID and Order_Date are mapped.
- Square brackets contain minimum and maximum occurrences of an element.
- An optional field is indicated by [0,1]. In this example, First_Class is optional.
- A repeating field is indicated by [minoccurs, maxoccurs].
- A choice field is indicated by a choice line; under the choice line are the possible choices. In this example, First_Class, Second_Class, and Airmail are choices of Delivery_Method.
- The type of each element is indicated in round brackets after the element name.
- If the message schema uses namespaces, the namespace prefix is shown before the element name, separated by a colon.

Use the Source pane to invoke a number of actions, a list of which is displayed when you right-click within the Source pane. The following table describes the available actions.

Action	Description	Related tasks
Undo	Undo previous action	
Redo	Redo previous action	
Revert	Discard	

Action	Description	Related tasks
Open Declaration (message)	Display the element definition from the message set. For this action to be available, select any source message element except LocalEnvironment or Headers.	
Open Declaration (database)	Display the database, schema, or table definition from the database. For this action to be available, select any source database object.	
Show Derived Types	Hide or display derived types for an element in the source or target pane. For this action to be available, select a target element displayed as a specialization folder in the source pane.	
Show Substituting elements	Hide or display the substituting elements of the head element in the source or target pane. For this action to be available, select a target element displayed as a substitutions folder in the source pane.	
Add Sources	Add a message definition or a database table to a source. For this action to be available, select any source object.	"Adding messages or message components to the source or target" on page 575, "Adding a database as a source or target" on page 575
Go To	For this action to be available, select any source object.	
Delete (message)	Remove a message and any existing maps from the source. For this action to be available, select the source message root ($source).	

Action	Description	Related tasks
Delete (database)	Remove a database and any existing maps from the source. For this action to be available, select the source database root ($db:select).	
Delete (database stored procedure)	Remove a database stored procedure and any existing maps from the source. For this action to be available, select the database stored procedure root ($db:proc).	
Delete (database user-defined function)	Remove a database user-defined function and any existing maps from the source. For this action to be available, select the database user-defined function root ($db:func).	
Map from Source	Create a map between the focus source element and the focus target element. For this action to be available, select compatible source and target elements.	"Mapping a target element from source message elements" on page 564, "Mapping from source: by selection" on page 555
Map by Name	Create a map between the focus source element and the focus target element. For this action to be available, select compatible source and target elements.	"Mapping a target element from source message elements" on page 564, "Mapping from source: by name" on page 555
Accumulate	If the source and target fields contain numeric data types, this action maps all occurrences of a repeating source field to a non-repeating target, resulting in the sum of all the source elements. For this action to be available, select the source and target element.	"Configuring a repeating source and a non-repeating target" on page 571
Create New Submap	For this action to be available, select source and target elements that are either elements of complex types or wildcard elements.	"Creating and calling submaps and subroutines" on page 590, "Creating a new submap" on page 590, "Creating a new submap for a wildcard source" on page 591

Action	Description	Related tasks
Create New Database Submap	Create a submap to modify a database	"Creating a submap to modify a database" on page 592
Call Existing Submap	Call an existing submap	"Creating and calling submaps and subroutines" on page 590, "Calling a submap" on page 594
Call ESQL Routine	Call an ESQL routine	"Creating and calling submaps and subroutines" on page 590, "Calling an ESQL routine" on page 596
Call Java Method	Call a Java Method	"Calling a Java method" on page 597
Add or Remove Headers and Folders	Include message headers and folders for source messages in a message map	"Mapping headers and folders" on page 573
Toggle Add/Remove Stored Procedure Return Value	Specify if a database stored procedure sets a return value. DB2 on z/OS and Oracle stored procedures do not set a return value.	"Mapping a target element from database stored procedures" on page 585
Add or Remove Result Set Columns	Specify the Result Set Columns for a database stored procedure	"Mapping a target element from database stored procedures" on page 585
Save	Save the .msgmap file	

Message Mapping editor Target pane

Details of the elements present in the Target pane of the Message Mapping Editor.

The following example shows the "Message Mapping editor" on page 1585. The pane that is labelled as 2 in the example is the Target pane:



The following list describes the elements that are present in the Target pane:

- A target message is identified by $target.
- A mapped entry is indicated by a yellow triangle alongside the element. In this example, Customer_ID, Order_Number, and Order_Date are mapped.
- Square brackets contain minimum and maximum occurrences of an element.
- An optional field is indicated by [0,1]. In this example, First_Class is optional.
- A repeating field is indicated by [minoccurs, maxoccurs].
- A choice field is indicated by a choice line; under the choice line are the possible choices. In this example, First_Class, Second_Class, and Airmail are choices of Delivery_Method.
- The type of each element is indicated in round brackets after the element name.
- If the message schema uses namespaces, the namespace prefix is shown before the element name, separated by a colon.

Use the Target pane to invoke a number of actions, a list of which is displayed when you right-click within the Target pane. The following table describes the available actions.

Action	Description	Related tasks
Undo	Undo previous action	
Redo	Redo previous action	
Revert	Discard	

Action	Description	Related tasks
Open Declaration (message)	Display the element definition from the message set. For this action to be available, select any target message element except LocalEnvironment or Headers.	
Open Declaration (database)	Display the database, schema, or table definition from the database. For this action to be available, select any target database object.	
Show Derived Types	Hide or display derived types for an element in the source or target pane. For this action to be available, select a target element displayed as a specialization folder in the target pane.	
Show Substituting elements	Hide or display the substituting elements of the head element in the source or target pane. For this action to be available, select a target element displayed as a substitutions folder in the target pane.	
Add Sources and Targets	Add a message definition or a database table to a source. For this action to be available, select any target object.	"Adding messages or message components to the source or target" on page 575, "Adding a database as a source or target" on page 575
Go To	For this action to be available, select any target object.	
Delete (message)	Remove a message and any existing maps from the source. For this action to be available, select the target message root ($target).	

Action	Description	Related tasks
Map from Source	Create a map between the focus source element and the focus target element. For this action to be available, select compatible source and target elements.	"Mapping a target element from source message elements" on page 564, "Mapping from source: by selection" on page 555
Map by Name	Create a map between the focus source element and the focus target element. For this action to be available, select compatible source and target elements.	"Mapping a target element from source message elements" on page 564, "Mapping from source: by name" on page 555
Enter Expression	For this action to be available, select any target object except $target	"Setting the value of a target element to a constant" on page 566, "Setting the value of a target element using an expression or function" on page 568
Accumulate	If the source and target fields contain numeric data types, this action maps all occurrences of a repeating source field to a non-repeating target, resulting in the sum of all the source elements. For this action to be available, select the source and target element.	"Configuring a repeating source and a non-repeating target" on page 571
Create New Submap	For this action to be available, select source and target elements that are either elements of complex types or wildcard elements.	"Creating and calling submaps and subroutines" on page 590, "Creating a new submap" on page 590, "Creating a new submap for a wildcard source" on page 591
Call Existing Submap	Call an existing submap	"Creating and calling submaps and subroutines" on page 590, "Calling a submap" on page 594
Call ESQL Routine	Call an existing ESQL routine	"Creating and calling submaps and subroutines" on page 590, "Calling an ESQL routine" on page 596
Save	Save the .msgmap file	

Message Mapping editor Edit pane

Details of how you use the Edit pane of the Message Mapping Editor.

The following example shows the “Message Mapping editor” on page 1585. The pane that is labelled as 3 in the example is the Edit pane:



When you have selected a source or target element, use the Edit pane to enter an expression. Right-click inside the Edit pane to invoke a list of available actions, most of which are standard Windows functions, such as cut, copy, and paste. Click **Edit** → **Content Assist** (or press Ctrl+Space) to access ESQL Content Assist, which provides a drop-down list of functions that are available in a Mapping node.

To display the definition associated with a selected element or database object, right-click in the Edit pane, and click **Open Declaration**. The appropriate editor opens to display the definition associated with the element or database definition.

Message Mapping editor Spreadsheet pane

Details of the actions available to you in the Spreadsheet pane of the Message Mapping Editor.

The following example shows the “Message Mapping editor” on page 1585. The pane that is labelled as 4 in the example is the Spreadsheet pane:



Use the Spreadsheet pane to invoke a number of actions, a list of which is displayed when you right-click within the Spreadsheet pane. The following table describes the available actions.

Action	Description	Related tasks
Undo	Undo previous action	
Redo	Redo previous action	
Revert	Discard	
Open Declaration (message)	Display the element definition from the message set. For this action to be available, select any message element except LocalEnvironment or Headers.	
Open Declaration (database)	Display the database, schema, or table definition from the database. For this action to be available, select any database object.	
Add Sources and Targets	Add a message definition to a target.	"Adding messages or message components to the source or target" on page 575, "Adding a database as a source or target" on page 575
Copy	Copy the selected item to the clipboard.	

Action	Description	Related tasks
Paste	Paste the item from the clipboard.	
Delete	Remove a row from the Spreadsheet.	
For	Define a repeating condition.	"Configuring a repeating source and a non-repeating target" on page 571, "Configuring a repeating source and a repeating target" on page 572
If	Define what must evaluate to 'true' to process subsequent mappings.	"Configuring a repeating source and a non-repeating target" on page 571, "Configuring conditional mappings" on page 570
ElseIf	Define what must evaluate to 'true' to process subsequent mappings if previous If or Elseif does not evaluate to 'true'..	"Configuring a repeating source and a non-repeating target" on page 571, "Configuring conditional mappings" on page 570
Else	Placeholder to process subsequent mappings if previous If or ElseIf does not evaluate to 'true'.	"Configuring conditional mappings" on page 570
Select Data Source	Define a database to be used in the mapping.	
Insert Children	Expand a structure so that each of its children has a row in the spreadsheet.	
Insert Sibling After	Create a number of new rows in the spreadsheet to set the values of specific instances of a repeating field. Can also be used to insert any non-repeating element, attribute or database column if valid at the selected location.	"Configuring a non-repeating source and a repeating target" on page 572
Insert Sibling Before	Create a number of new rows in the spreadsheet to set the values of specific instances of a repeating field. Can also be used to insert any non-repeating element, attribute or database column if valid at the selected location.	"Configuring a non-repeating source and a repeating target" on page 572
Replace	Substitute an element, attribute or database column in the spreadsheet with a similar item, retaining the mapping expression and any child mapping statements.	
Save	Save the .msgmap file.	

Mapping node

The Mapping node has one or more mappings that are stored in message map files (with a `.msgmap` file extension). These files are configured using the "Message Mapping editor" on page 1585.

A Mapping node must contain the following inputs and outputs:

- Zero or one source (input) messages
- Zero or more source (input) databases
- One or more target (output) messages

You must define, in message definition files in a message set, the source and target messages that are to be mapped. You can specify the parser of the source message at run time (for example, in an MQRFH2 header), but the target message is built using the runtime parser that is specified by the *Message Domain* property of the message set.

If a message mapping is between elements of different types, you might need to include casts in your mapping definitions, depending on which runtime parser is specified by the *Message Domain* property of your message set.

The Mapping node uses a language to manipulate messages that are based on XPath.

To develop message mappings for a Mapping node, use the Message Mapping editor, which provides separate panes for working with sources, targets and expressions.

Mapping node syntax

In a Mapping node, the source message, if present, is identified in the "Message Mapping editor" on page 1585 by $source.

The message tree is represented in XPath format. For example, if you have an element called Body within a source message called Envelope, this is represented in the Mapping node as:

```
$source/soap11:Envelope/soap11:Body
```

where *soap11* is a namespace prefix.

The first target message is identified by $target; additional target messages are identified by $target_1, $target_2, and so on.

The first source database is identified by $db:select; additional source databases are identified by $db:select_1, $db:select_2, and so on.

The first source database stored procedure is identified by $db:proc; additional source stored procedures are identified by $db:proc_1, $db:proc_2, and so on.

The first source database user-defined function is identified by $db:func; additional source user-defined functions are identified by $db:func_1, $db:func_2, and so on.

The database element is represented in the following format:

```
$db:select.DB.SCH.TAB.COL1
```

where:

DB is the database name

SCH is the database schema name

TAB is the table name

COL1 is the column name

You can also use the Mapping node to:

- make comparisons
- perform arithmetic
- create complex conditions

The comparison operators are:

= equals

!= not equals

> greater than

>= greater than or equals

< less than

<= less than or equals

The arithmetic operators are:

+ plus

- minus

* multiply

div divide

Conditional operators ‘or' and ‘and' are supported (these are case-sensitive).

The following objects can be mapped:

- Local Environment
  - Destination
  - WrittenDestination
  - File
  - SOAP
  - TCPIP
  - ServiceRegistry
  - Adapter
  - Wildcard
  - Variables
- Message headers (optional)
  - MQ Headers
  - HTTP Headers
  - JMSTransport
  - E-mail Headers
- Message elements
- Database columns

Database objects with names that do not conform to the XML NCName format

Some database objects have names that do not conform to the XML NCName format (for example, the names contains characters like '#', or '$'). To reference such database objects use the `msgmap:db-path` function. For more information, see "Predefined mapping functions" on page 1606

Mapping node functions

You can configure your message mappings to use a variety of predefined and user-defined functions.

The following predefined functions are available to use in your message maps:

- ESQL - prefixed esql:
- XPath - prefixed fn:
- Mapping - prefixed msgmap:
- Schema casts - prefixed xs:

Not all ESQL functions can be used in a Mapping node. For information about which functions are supported, and for a description of how to achieve equivalent processing for ESQL functions that are not supported, see the ESQL topics. For information about the predefined ESQL functions, see "Predefined ESQL mapping functions."

The fn:true() function (which always returns true) and the fn:false() function (which always returns false) are examples of XPath functions. You can get more information about the other XPath functions and XPath syntax from the online W3C XML Path Language document. For information about the predefined XPath functions, see "Predefined XPath mapping functions" on page 1602.

For information about the predefined mapping functions, see "Predefined mapping functions" on page 1606. See "Mapping node casts" on page 1608 for a list of the schema casts.

The Mapping node can also:

- Set the value of a target to a WebSphere MQ constant. The expression to set the value looks similar to a function with $mq: used as the prefix.
- Call a Java method directly. The expression to set the value looks similar to a function with java: used as a prefix.

Predefined ESQL mapping functions

A table of predefined ESQL functions for use with message maps.

This table details the predefined ESQL mapping functions that are available to use with message maps:

Name	ESQL equivalent	Notes
Numeric functions: abs absval acos asin atan atan2 bitand bitnot bitor bitxor ceil ceiling cos cosh cot degrees exp floor in log log10 mod power radians rand sign sin sinh sqrt tan tanh truncate	ESQL function of the name same name such as ABS and ABSVAL.	The same parameters apply as for ESQL.

Name	ESQL equivalent	Notes
String functions: left length lower lcase ltrim replace replicate right rtrim space translate upper ucase	ESQL function of the same name such as LEFT and LENGTH.	The same parameters apply as for ESQL.
Field functions: bitstream fieldname fieldnamespace fieldtype fieldvalue lastmove samefield	ESQL function of the same name such as BITSTREAM and FIELDNAME.	The same parameters apply as for ESQL.
asbitstream	These signatures are supported: ASBITSTREAM(*FieldRef*) ASBITSTREAM(*FieldRef*, *typeExp*, *setExp*, *formatExp*) ASBITSTREAM(*FieldRef*, *typeExp*, *setExp*, *formatExp*, *encodingExp*, *ccsidExp*) ASBITSTREAM(*FieldRef*, *typeExp*, *setExp*, *formatExp*, *encodingExp*, *ccsidExp*, *options*)	*FieldRef* is a source field reference such as $source/po:PurchaseOrder *typeExp* is a string literal of the name of the message body, such as purchaseOrder, optionally qualified with a namespace URI, such as {http://www.ibm.com}:purchaseOrder *setExp* is a string literal of the name of the message set, such as PurchaseOrder *formatExp* is a string literal of the wire format of the message, such as XML1 *encodingExp* and *ccsidExp* evaluate to integers with values corresponding to ESQL ENCODING and CCSID constants. *options* is an ESQL constant or bit-or of ESQL constant that evaluates to an integer.
cardinality	CARDINALITY	The same parameters apply as for ESQL.
coalesce	COALESCE	The same parameters apply as for ESQL.
current-date	CURRENT_DATE	No parameters apply.
current-gmtdate	CURRENT_GMTDATE	No parameters apply.
current-gmttime	CURRENT_GMTTIME	No parameters apply.
current-gmttimestamp	CURRENT_ GMTTIMESTAMP	No parameters apply.
current-time	CURRENT_TIME	No parameters apply.
current-timestamp	CURRENT_TIMESTAMP	No parameters apply.
date	DATE	
for	FOR (expression)	Optional parameters are not supported.
gmttime	GMTTIME	
gmttimestamp	GMTTIMESTAMP	

interval-year	INTERVAL YEAR	The same parameters apply as for ESQL. Some examples:
		
		`esql:interval-minute('90')`
		`esql:interval-year-to-month('1-06')`

interval-year-to-month	INTERVAL YEAR TO MONTH	
interval-month	INTERVAL MONTH	
interval-day	INTERVAL DAY	
interval-day-to-hour	INTERVAL DAY TO HOUR	
interval-day-to-minute	INTERVAL DAY TO MINUTE	
interval-day-to-second	INTERVAL DAY TO SECOND	
interval-hour	INTERVAL HOUR	
interval-hour-to-minute	INTERVAL HOUR TO MINUTE	
interval-hour-to-second	INTERVAL HOUR TO SECOND	
interval-minute	INTERVAL MINUTE	
interval-minute-to-second	INTERVAL MINUTE TO SECOND	
interval-second	INTERVAL SECOND	

is-null	Operand IS NULL	Some examples:
		
		`esql:is-null($source/po:purchaseOrder/po:comment)`
		`esql:is-null`
		`($db:select.ACME.PARTS.INVENTORY.LAST_TRANSACTION)`

Name	ESQL equivalent	Notes
like	source LIKE pattern	For example: `esql:like ($source/po:purchaseOrder/shipTo/first_name,'Fred')`
	source LIKE pattern ESCAPE EscapeChar	For example: `esql:like ($source/po:purchaseOrder/shipTo /zip,'L6F$_1C7','$')`
local-timezone	LOCAL_TIMEZONE	
nullif	NULLIF	The same parameters apply as for ESQL.
overlay	OVERLAY Str1 PLACING Str2 FROM Start	For example: `esql:overlay ($source/po:purchaseOrder/shipTo/city,'abc',2)`
	OVERLAY Str1 PLACING Str2 FROM Start For Length	For example: `esql:overlay ($source/po:purchaseOrder/shipTo/city,'abcde',2,3)`
position	POSITION searchExp IN SourceExp	For example: `esql:position ('aet',$source/po:purchaseOrder/shipTo/first_name)`
	POSITION searchExp IN SourceExp FROM FromExp	For example: `esql:position ('do',$source/po:purchaseOrder/shipTo/last_name,1)`
	POSITION searchExp IN SourceExp FROM FromExp REPEAT RepeatExp	For example: `esql:position ('a',$source/po:purchaseOrder/billTo /first_name,1,2)`
round	ROUND	Optional parameters are not supported.
sqlcode	SQLCODE	No parameters apply.
sqlerrortext	SQLERRORTEXT	
sqlnativeerror	SQLNATIVEERROR	
sqlstate	SQLSTATE	
time	TIME	
timestamp	TIMESTAMP	The same parameters apply as for ESQL. For example: `esql:gmttimestamp ('1999-12-31 23:59:59.999999')`
trim-leading	TRIM LEADING FROM Source	For example: `esql:trim-leading ($source/po:purchaseOrder/shipTo/state)`
	TRIM LEADING Singleton FROM Source	For example: `esql:trim-leading ('G',$source/po:purchaseOrder/shipTo/zip)`
trim-trailing	TRIM TRAILING FROM Source	For example: `esql:trim-trailing ($source/po:purchaseOrder/billTo/last_name)`
	TRIM TRAILING Singleton FROM Source	For example: `esql:trim-trailing ('e',$source/po:purchaseOrder/billTo/street)`

Name	ESQL equivalent	Notes
trim-both	TRIM BOTH FROM Source	For example: `esql:trim-both ($source/po:purchaseOrder/shipTo/city)`
	TRIM BOTH Singleton FROM Source	For example: `esql:trim-both (",$source/po:purchaseOrder/shipTo/city)`
trim	TRIM Source	For example: `esql:trim ($source/po:purchaseOrder/shipTo/city)`
	TRIM Singleton FROM Source	For example: `esql:trim (",$source/po:purchaseOrder/shipTo/city)`
uuidasblob	UUIDASBLOB	Takes zero or more parameters as in ESQL.
uuidaschar	UUIDASCHAR	

Predefined XPath mapping functions

A table of predefined XPath functions for use with message maps.

This table details the predefined XPath functions that are available for use with message maps. You can get more information about XPath functions and XPath syntax from the online W3C XML Path Language document.

Name	Parameters	Notes
true		
false		
sum	Source field from the message or database or “XPath "for expression"” on page 1605.	Source supports an XPath predicate. An XPath predicate is an expression enclosed in square brackets, serving to filter a sequence, retaining some items and discarding others. Predicates are supported on the XPath aggregate functions of `avg`, `count`, `max`, `min`, and `sum`. Predicates must be in one of the two forms: • An integer literal – aggregation is done for elements whose context position equals the integer. • A Boolean expression – aggregation is done for elements that make the Boolean expression evaluate to true. See “Aggregating XPath expressions conditionally” on page 1603 for more information.
avg		
max		
min		
count		
concat	Two or more strings.	You cannot use `fn:concat($source/myElem)` to concatenate instances of 'myElem'.
not	1- Expression resolved to a Boolean value.	
exists	Source field from the message or database.	
empty		

Name	Parameters	Notes
substring	1- String 2- Zero-bases starting index 3- Length	For example: `fn:substring ($source/po:purchaseOrder/billTo/street, 3, 5)`
substring-before	Two strings	Returns the substring of the first string that precedes the first occurrence of the second string.
substring-after	Two strings	Returns the substring of the first string that follows the first occurrence of the second string.
starts-with	Two strings	Returns a Boolean value indicating whether the first string starts with the second string.
ends-with	Two strings	Returns a Boolean value indicating whether the first string ends with the second string.
contains	Two strings	Returns a Boolean value indicating whether the first string contains the second string.
year-from-dateTime	1- xs:dateTime	For example: `fn:month-from-dateTime (xs:dateTime($source/po:purchaseOrder /shipTo/datetime))` where `$source/po:purchaseOrder/shipTo/datetime` is xs:string.
month-from-dateTime		
day-from-dateTime		
hours-from-dateTime		
minutes-from-dateTime		
seconds-from-dateTime		
year-from-date	1-xs:date	For example: `fn:year-from-date(xs:date ($source/po:purchaseOrder/billTo/date))` where `$source/po:purchaseOrder/billTo/date` is xs:string.
month-from-date		
day-from-date		

hours-from-time	1- xs:time	Some examples:
		
		`fn:hours-from-time(xs:time("13:20:10:5"))`
		`fn:hours-from-time(xs:time ($source/po:purchaseOrder/shipTo/time))`

minutes-from-time		
seconds-from-time		
years-from-duration	1- xdt:dayTimeDuration	For example: `fn:minutes-from-duration (xdt:dayTimeDuration(PT47H30M))`
months-from-duration		
days-from-duration		
hours-from-duration		
minutes-from-duration		
seconds-from-duration		

Aggregating XPath expressions conditionally:

You can perform calculations using conditions, called predicates in XPath, on the aggregate functions in the Mapping node. The aggregate functions are `avg`, `count`, `max`, `min`, and `sum`.

XPath aggregate functions

XPath aggregate functions take a sequence as their argument:

```
fn:count($arg as item()*) as xs:integer
fn:avg($arg as xs:anyAtomicType*) as xs:anyAtomicType?
fn:max($arg as xs:anyAtomicType*) as xs:anyAtomicType?
fn:min($arg as xs:anyAtomicType*) as xs:anyAtomicType?
fn:sum($arg as xs:anyAtomicType*) as xs:anyAtomicType?
```

where the atomic value can be, for example, an integer, a string, or a Boolean value.

In an aggregation

```
fn:sum($source/inventory/category/product/price)
```

the node

```
$source/inventory/category/product/price
```

is put into its typed value, which is a sequence of `Atomictype` `price` values.

In the following aggregation, the argument is a node `tns:price`, which is contained in `tns:product`, which is in turn contained in `tns:category`, and so on.

```
fn:sum($source/tns:Inventory/tns:category/tns:product/tns:price)
```

How aggregation works in this sequence depends upon the scope of the iteration (or `for`) loop for `product` and `category`. For example, if there is no iteration loop, summation is done for all instances of all `prices` in all `products` in all `categories`.

However, if there is an iteration loop on `category`, one summation is calculated for each `category`. The summation is obtained from all `prices` of all `products` of a particular `category` being iterated upon.

You can add a predicate (see predicates for further information) to make the result more specific. For example:

```
fn:sum($source/tns:Inventory/tns:category[$source/tns:Inventory/tns:category/
  tns:c_id='abc' ]/tns:product[3]/tns:price)
```

has a predicate of `tns:c_id='abc'` on `category`, and a predicate of `3` on `product`. The result that you obtain is the third `product` in the `category` that has `tns:c_id='abc'`.

In both of the preceding examples, whether the expression contains a condition or not, you reference one source item only; it is `price` that is aggregated.

Predicates are supported only when they are used in message sources. For example, the `[ boolean expression ]` must be used within a segment of a path that represents a message source. Predicates are not supported in `$db:select` or `$db:proc`.

Consider a more complicated scenario, where product prices are kept in a database table, and the input message contains the quantity of the product being ordered. The objective is to calculate a total price by adding up all prices multiplied by quantity.

It might seem to be straightforward to write the following aggregation function:

```
fn:sum($source/inventory/product/quantity × $db:select/PRICE_TB/PRICE)
```

However, the first step of evaluating an XPath arithmetic expression is to evaluate its operands. If any operand is evaluated into a sequence of more than one item,

either only the first item is used in the arithmetic expression, or an error is raised; for further information, see Arithmetic expressions.

You, therefore, cannot add up the product of `quantity` and `PRICE` for each `product`. If you want to aggregate the result of an arithmetic expression, see “XPath "for expression".”

XPath "for expression":

You can use the "for expression" to perform specific calculations as the argument of an aggregation function in the Mapping node.

For expressions

To perform a specific operation you can use the XPath `for expression` iteration facility; see for expression for further information.

You can express a sequence of price multiplied by quantity in many ways using the `for expression`. For example:

- `for $i in $source/inventory/category/product return $i/price * $i/quantity`

  You can use a similar expression when all operands of the return expression (`price` and `quantity`) are defined in the same repeatable container (`product`). Only one variable is needed in the `for expression`.
- ```
  for $i in $db:select, $j in $source/inventory/category/product
  [$i.db1.sch2.PRICE_TB.PROD_ID=$j/product_id]
  return $i.db1.sch2.PRICE_TB.PRICE * $j/quantity
  ```

  You can use a similar expression when some operands of the return expression (`price`) are kept in a database table, and other operands (`quantity`) are kept in a message. At least one variable is needed for the path to a database result set, and another variable is needed for the path to a repeatable XML element.
- `for $i in $source/order_info/product1, $j in $source/price_record/product2 [$i/product_id=$j/product_id] return $i/quantity * $j/price`

  You can use a similar expression when the operands (`price` and `quantity`) are defined in different repeatable elements (`product1` and `product2`) in the source message.

When you have an expression that represents a sequence of items, aggregation can be done by using the `for expression` as the argument of the aggregation function.

Examples

Obtain the average cost - that is, price multiplied by quantity - for all products:

```
fn:avg(for $i in $source/inventory/category/product
   return $i/price * $i/quantity)
```

Obtain the total cost of all products where prices are from a database, and quantities are from a message and paired up based on product identifier:

```
fn:sum(for $i in $db:select, $j in $source/inventory/category/product
   [$i.db1.sch2.PRICE_TB.PROD_ID=$j/product_id]
   return $i.db1.sch2.PRICE_TB.PRICE * $j/quantity)
```

Obtain the length of the longest product identifier text:

```
fn:max(for $i in $source/inventory/category/product
   return esql:length($i/id))
```
```

Obtain the minimum price when the price was stored in a message as a string, and convert each source item into a numeric value before using an aggregate function:

```
fn:min(for $i in $source/inventory/category/product
   return xs:decimal($i/price))
```

The aggregation functions that are supported in WebSphere Message Broker Version 6.1.0.2, and earlier, can be expressed using a `for expression`.

For example, in WebSphere Message Broker Version 6.1.0.2 you can have:

```
fn:sum($source/inventory/category/product/price)
```

and this format is still supported. This expression is equivalent to:

```
fn:sum(for $i in $source/inventory/category/product
   return $i/price)
```

Predefined mapping functions

Some predefined mapping functions are provided for use with message maps.

This table details the predefined mapping functions that are available to use with message maps.

Name	Parameters	Return	Notes
cdata-element	One string	Nothing	Create an XML element with CData content in the following target message domains: • XMLNSC • SOAP • XMLNS • XML • JMSMap • JMSStream For example: `msgmap:cdata-element('<date><month>05 </month><day>11</day><year>2008</year></date>')`

db-path	These signatures are supported:
	msgmap:db-path(*databasePath*)
	msgmap:db-path(*databasePath*, *delimiter*)
		Nothing	Used when the database path does not conform to the XML NCName format. For example, the database path contains characters like '#', or '$'.
			databasePath is a reference to either a `$db:select` statement, a `$db:insert` statement, a `$db:update` statement, a `$db:delete` statement, a `$db:proc` statement, or a `$db:func` statement.
			delimiter is a string literal used between the segments in the database path. *delimiter* has a default value of '.'.
			For example:
			`msgmap:db-path("$db:select.DB#1.SCH$#1. CustInfo.Name")`
			In this case, the default value, '.', is used as the delimiter.
			`msgmap:db-path("$db:select/DB#1/SCH$#1/ Cust.Info/Name", "/")`
			In this case, '/' is used as the delimiter because the database table name includes the default value.

Name	Parameters	Return	Notes

occurrence	Source field from the message or database	The index of the source field.	Often used in a condition statement when source repeats to execute specific statements for a specific occurrence. For example:
			`msgmap:occurrence ($source/po:purchaseOrder /items)=2`
			means the second field,
			`po:purchaseOrder`
			is being processed. Use code similar to the previous example if you want to specify a particular source occurrence. If you only want to assign the index value to a target you can specify the following code:
			`msgmap:occurrence ($source/po:purchaseOrder /items)`

exact-type	1- Source field from the message or database
	2- Namespace prefix
	3- Name of the type
		True if the source is of the specified type in the specified namespace.	Often used in a condition to execute specific statements for a specific source type. For example:
			`msgmap:exact-type ($source/tn1:msg2,'tn1', 'extendedMsgType')`
			The namespace prefix can be '*', indicating that only the name of the type is to be checked. For example:
			`msgmap:exact-type($source/tn1:msg2,'*', 'extendedMsgType')`
			returns true if the element
			`$source/tn1:msg2`
			has a type of
			`extendedMsgType`
			in any namespace.

empty-element()	None	Nothing	Creates an XML element with an empty tag. For example, if an element is named MyElement and the mapping expression for MyElement is set to msgmap:empty-element(), the output message will contain an element with no content: `<MyElement/>` Call this function only for an XML element.

Name	Parameters	Return	Notes
element-from-bitstream	These signatures are supported: msgmap:element-from-bitstream(*StreamRef*) msgmap:element-from-bitstream(*StreamRef*, *typeExp*, *setExp*, *formatExp*) msgmap:element-from-bitstream(*StreamRef*, *typeExp*, *setExp*, *formatExp*, *encodingExp*, *ccsidExp*) msgmap:element-from-bitstream(*StreamRef*, *typeExp*, *setExp*, *formatExp*, *encodingExp*, *ccsidExp*, *options*)	Nothing	Used to parse a bit stream. This function can be called only for a message element target. The parsed bit stream is placed in the target message tree as the target element. *StreamRef* is a reference to a BLOB of stream, such as $source/BLOB or $db:select.dsn.schema.table.column *typeExp* is a string literal of the name of the message body, such as purchaseOrder, optionally qualified with a namespace URI, such as {http://www.ibm.com}:purchaseOrder *setExp* is a string literal of the name of the message set, such as PurchaseOrder *formatExp* is a string literal of the wire format of the message, such as XML1 *encodingExp* and *ccsidExp* evaluate to integers with values corresponding to ESQL ENCODING and CCSID constants *options* is an ESQL constant or bit-or of ESQL constants that evaluate to an integer.

Mapping node casts

Source and target elements can be of different types in a Mapping node.

Depending on which runtime parsers are used, automatic casting cannot be done. In these cases, use one of the cast functions shown in the following table.

Function name	Signature	Parameter type
xs:anyURI	xs:anyURI(*&exp*) xs:anyURI(*&exp*, *&ccsid*) xs:anyURI(*&exp*, *&ccsid*, *&encoding*)	*$exp* is a reference to a source element of one of the following types: • xs:base64Binary • xs:boolean • xs:date • xs:dateTime • xs:dayTimeDuration • xs:decimal • xs:double • xs:duration • xs:float • xs:gDay • xs:gMonth • xs:gMonthDay • xs:gYear • xs:gYearMonth • xs:hexBinary • xs:int • xs:integer • xs:long • xs:QName • xs:string • xs:time • xs:yearMonthDuration *&ccsid* is one of the following values: • $source/Properties/CodedCharSetId • $source/MQMD/CodedCharSetId • An integer. For example, 1208 for UTF-8 *&encoding* is one of the following values: • $source/Properties/Encoding • $source/MQMD/Encoding • $mq:MQENC_WINDOWS (546) • $mq:MQENC_UNIX (273) • $mq:MQENC_390 (785) • Other $mq:MQENC_* constants and their BIT OR

Function name	Signature	Parameter type
xs:base64Binary	xs:base64Binary(*&exp*) xs:base64Binary(*&exp*, *&ccsid*) xs:base64Binary(*&exp*, *&ccsid*, *&encoding*)	*$exp* is a reference to a source element of one of the following types: • xs:hexBinary • xs:int • xs:string *&ccsid* is one of the following values: • $source/Properties/CodedCharSetId • $source/MQMD/CodedCharSetId • An integer. For example, 1208 for UTF-8 *&encoding* is one of the following values: • $source/Properties/Encoding • $source/MQMD/Encoding • $mq:MQENC_WINDOWS (546) • $mq:MQENC_UNIX (273) • $mq:MQENC_390 (785) • Other $mq:MQENC_* constants and their BIT OR
xs:boolean	xs:boolean(*&exp*)	*$exp* is a reference to a source element of type xs:string.
xs:date	xs:date(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:dateTime • xs:gDay • xs:gMonth • xs:gMonthDay • xs:gYear • xs:gYearMonth • xs:string
xs:dateTime	xs:dateTime(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:date • xs:string • xs:time
xs:dayTimeDuration	xs:dayTimeDuration(*&exp*)	*$exp* is a reference to a source element of type xs:string.

Function name	Signature	Parameter type
xs:decimal	xs:decimal(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:decimal • xs:float • xs:int • xs:integer • xs:string
xs:double	xs:double(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:duration • xs:float • xs:int • xs:string
xs:duration	xs:duration(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:decimal • xs:float • xs:int • xs:string
xs:float	xs:float(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:double • xs:duration • xs:int • xs:string
xs:gDay	xs:gDay(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:date • xs:dateTime • xs:gMonth • xs:gMonthDay • xs:gYear • xs:gYearMonth • xs:string

Function name	Signature	Parameter type
xs:gMonth	xs:gMonth(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:date • xs:dateTime • xs:gDay • xs:gMonthDay • xs:gYear • xs:gYearMonth • xs:string
xs:gMonthDay	xs:gMonthDay(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:date • xs:dateTime • xs:gDay • xs:gMonth • xs:gYear • xs:gYearMonth • xs:string
xs:gYear	xs:gYear(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:date • xs:dateTime • xs:gDay • xs:gMonth • xs:gMonthDay • xs:gYearMonth • xs:string
xs:gYearMonth	xs:gYearMonth(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:date • xs:dateTime • xs:gDay • xs:gMonth • xs:gMonthDay • xs:gYear • xs:string

Function name	Signature	Parameter type
xs:hexBinary	xs:hexBinary(*&exp*) xs:hexBinary(*&exp*, *&ccsid*) xs:hexBinary(*&exp*, *&ccsid*, *&encoding*)	*$exp* is a reference to a source element of one of the following types: • xs:base64Binary • xs:int • xs:string *&ccsid* is one of the following values: • $source/Properties/CodedCharSetId • $source/MQMD/CodedCharSetId • An integer. For example, 1208 for UTF-8 *&encoding* is one of the following values: • $source/Properties/Encoding • $source/MQMD/Encoding • $mq:MQENC_WINDOWS (546) • $mq:MQENC_UNIX (273) • $mq:MQENC_390 (785) • Other $mq:MQENC_* constants and their BIT OR
xs:int	xs:int(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:base64Binary • xs:decimal • xs:duration • xs:float • xs:hexBinary • xs:string
xs:integer	xs:integer(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:base64Binary • xs:decimal • xs:duration • xs:int • xs:string

Function name	Signature	Parameter type
xs:long	xs:long(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:base64Binary • xs:decimal • xs:duration • xs:float • xs:hexBinary • xs:string

Function name	Signature	Parameter type
xs:QName	xs:QName(*&exp*) xs:QName(*&exp*, *&ccsid*) xs:QName(*&exp*, *&ccsid*, *&encoding*)	*$exp* is a reference to a source element of one of the following types: • xs:anyURI • xs:base64Binary • xs:boolean • xs:date • xs:dateTime • xs:dayTimeDuration • xs:decimal • xs:double • xs:duration • xs:float • xs:gDay • xs:gMonth • xs:gMonthDay • xs:gYear • xs:gYearMonth • xs:hexBinary • xs:int • xs:integer • xs:long • xs:string • xs:time • xs:yearMonthDuration *&ccsid* is one of the following values: • $source/Properties/CodedCharSetId • $source/MQMD/CodedCharSetId • An integer. For example, 1208 for UTF-8 *&encoding* is one of the following values: • $source/Properties/Encoding • $source/MQMD/Encoding • $mq:MQENC_WINDOWS (546) • $mq:MQENC_UNIX (273) • $mq:MQENC_390 (785) • Other $mq:MQENC_* constants and their BIT OR

Function name	Signature	Parameter type
xs:string	xs:string(*&exp*) xs:string(*&exp*, *&ccsid*) xs:string(*&exp*, *&ccsid*, *&encoding*)	*$exp* is a reference to a source element of one of the following types: • xs:anyURI • xs:base64Binary • xs:boolean • xs:date • xs:dateTime • xs:dayTimeDuration • xs:decimal • xs:double • xs:duration • xs:float • xs:gDay • xs:gMonth • xs:gMonthDay • xs:gYear • xs:gYearMonth • xs:hexBinary • xs:int • xs:integer • xs:long • xs:QName • xs:time • xs:yearMonthDuration *&ccsid* is one of the following values: • $source/Properties/CodedCharSetId • $source/MQMD/CodedCharSetId • An integer. For example, 1208 for UTF-8 *&encoding* is one of the following values: • $source/Properties/Encoding • $source/MQMD/Encoding • $mq:MQENC_WINDOWS (546) • $mq:MQENC_UNIX (273) • $mq:MQENC_390 (785) • Other $mq:MQENC_* constants and their BIT OR
xs:time	xs:time(*&exp*)	*$exp* is a reference to a source element of one of the following types: • xs:dateTime • xs:string

Function name	Signature	Parameter type
xs:yearMonthDuration	xs:yearMonthDuration(*&exp*)	*$exp* is a reference to a source element of type xs:string.

Headers and Mapping node

This topic lists the headers that can be manipulated by the Mapping node.

You can map these headers:

- MQ Headers
  - MQMD
  - MQCFH header with root element MQPCF
  - MQCIH
  - MQDLH
  - MQIIH
  - MQMDE
  - MQRFH
  - MQRFH header with MQRFH2 or MQRFH2C parser
  - MQRMH
  - MQSAPH
  - MQWIH
  - SMQ_BMH
- Email Headers
  - EmailOutputHeader
- HTTP Headers
  - HTTPInputHeader
  - HTTPReplyHeader
  - HTTPRequestHeader
  - HTTPResponseHeader
- JMSTransport

Migrating message mappings from Version 5.0

Use the mqsimigratemfmaps command to migrate message mappings to the Version 6.1 format.

The mqsimigratemfmaps command creates Version 6.1 mapping files (`.msgmap`) from your Version 5.0 mapping files (`.mfmap`).

When you migrate message mappings from Version 5.0, read the restrictions that apply.

The following table lists the mapping functions that are supported in Version 5.0 but not supported in Version 6.1, and shows the error messages that you might see. Mappings that contain these Version 5.0 functions cannot be migrated to Version 6.1; you must re-create and redeploy these mappings using another node, such as a JavaCompute node. Alternatively, migrate as much of the mapping as possible using the migration command, view the error report to see details of the functions that could not be migrated, and create a new node that can execute those functions that were not migrated.

Supported in Version 5.0	Migration utility error message
Expressions that involve multiple instances of a repeating source element, for example: `src_msg.e[1] + src_msg.e[2] -> tgt_msg.e`	`Error:102: Unexpected index '2' encountered for target mappable 'e'. The expected index is '1'. Migration currently provides no support for expressions involving more than one instance of the same repeating-element.`
ESQL field references that contain the asterisk wildcard character "*". For example: `src_msg.e.* or src_msg.e.*[]`	`Error:130: ESQL field-reference 'src_msg.e.*' cannot be migrated. Migration currently provides no support for field-references containing '*'.`
Dynamic ESQL field references. For example: `src_msg.e.{'a' || 'b'}`	`Error:131: ESQL field-reference 'src_msg.e.{'a' || 'b'}' cannot be migrated. Migration currently provides no support for dynamic field-references.`
ESQL expressions that contain a reference to the temporary index-variable "#I". For example: `src_msg_e || "#I" -> tgt_msg.e`	`Error:128: ESQL expressions containing the variable '#I' anywhere other than the index of a repeating-element cannot be handled by the migration.`
Expressions within an index of a repeating element. For example: `src_msg.e[src_msg.a] or src_msg.e["#I" +5] or src_msg.e[< 3]`	`Error:116: ESQL field-reference 'src_msg.e[< 3]' cannot be migrated. Migration currently provides no support for indexes other than the variable '#I' and plain integer indexes.`
Aggregation functions MIN, MAX, and COUNT, used with the ESQL SELECT expression. For example: `SELECT MAX("#T".FIRSTNAME) FROM Database.CUSTOMER AS "#T" WHERE "#T".CUSTOMERID = 7`	`Error:135: The ESQL expression 'SELECT MAX("#T".FIRSTNAME) FROM Database.CUSTOMER AS "#T" WHERE "#T".CUSTOMERID = 7' could not be migrated. The expression contains syntax which has no direct equivalent in the new map-script language.`
ESQL IN operator. For example: `src_msg.e IN (1, 2, 3)`	`Error:135: The ESQL expression 'SELECT MAX("#T".FIRSTNAME) FROM Database.CUSTOMER AS "#T" WHERE "#T".CUSTOMERID = 7' could not be migrated.`

Restrictions on migrating message mappings

Learn how to migrate message maps from Version 5.0.

The programming model for message maps is different between Version 5.0 (where the file format is `.mfmap`) and Version 6.1 (where the format is `.msgmap`). Version 5.0 message maps have a procedural programming model, which is essentially an alternative ESQL, where you describe all the steps that are required to perform a transformation. Version 6.1 uses a declarative programming model, where you describe the result of the transformation, and the tools determine how to achieve that result.

Most migration failures result from message maps that contain too much information about the steps that perform the transformation, and not enough information about the required result. For these message maps, migration is enabled by changing the `.mfmap` file so that specific "how to" sections are separated into an ESQL function or procedure that can be called by the message map. The

.mfmap file calls the ESQL function instead of containing it as an expression. The mqsimigratemfmaps command then migrates the .mfmap file, but calls the ESQL function instead of logging a migration error.

A limitation is that ESQL (the run time for .mfmap and .msgmap files) cannot define functions that return complex element (or REFERENCE) values. The following procedure explains how to work around this complex element target limitation; in many cases, you must rewrite the map as an ESQL function. For more examples and information about calling ESQL from maps, see the following sample:

- Message Map

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

1. Determine whether you can define an ESQL function for the .mfmap file.
   a. When the target value is a complex element, or in ESQL terms a REFERENCE, the individual mapping must be rewritten in the .msgmap file. Delete the mapping from the .mfmap file, and proceed to Step 4.
   b. Use a function for all other cases: CHAR string, numbers, date, and time. Proceed to Step 2.
2. Determine the source parameters and returns type for your function.
   a. For each source path in the mapping, there must be one parameter in the function or procedure. For a function, all parameters are unchangeable. The type of the parameter must match the source data type.
   b. The function return type is the ESQL data type identified previously.
3. Update the .mfmap file to enable migration. Change the .mfmap file to invoke the function in the mapping, passing the source parameters to the function in the order in which they were listed in step 2a.
4. Re-run the mqsimigratemfmaps command to migrate the modified .mfmap file.
5. Repeat Steps 1 to 4 until no errors are reported in the migration log.
6. Start the Version 6.1 Message Broker Toolkit and open the migrated .msgmap file.
   a. For ESQL that is migrated as functions, there should be no errors.
   b. For complex element targets, rewrite the mapping by using the Version 6.1 features.

The following examples illustrate migration of .mfmap files to .msgmap files.

- To migrate a multiple reference to a repeating source expression:

  ```
  src_msg.e[1] + src_msg.e[2]
  ```

  compute the result in an ESQL function such as:

  ```
  CREATE FUNCTION addOneAndTwo(IN src_msg)
  BEGIN
   RETURN src_msg.e[1] + src_msg.e[2];
  END;
  ```

  In the .msgmap file, call the ESQL function addOneAndTwo by using the parent element **src_msg** as a parameter.

- An expression that does not use element names:

  ```
  src_msg.*
  ```

  or

  ```
  src_msg.*[]
  ```

can be processed using a function that takes the parent of the repeating field:

```
CREATE FUNCTION processAny(IN src_msg)
BEGIN
 DECLARE nodeRef REFERENCE TO src_msg.e.*;
 DECLARE result <dataType> <initialValue>;
 WHILE LASTMOVE nodeRef DO
  --expression goes here
  SET result = result;
 END WHILE;
 RETURN RESULT;
END;
```

In the `.msgmap` file, call the ESQL function processAny by using the parent element **src_msg** as a parameter.

- Expressions that dynamically compute element names:

```
src_msg.{'a' || 'b'}
```

can be processed by ESQL functions that process the parent of the repeating field:

```
CREATE FUNCTION processDynamicName(IN src_msg)
BEGIN
 RETURN src_msg.{'a' || 'b'};
END;
```

In the `.msgmap` file, call the ESQL function processDynamicName by using the parent element **src_msg** as a parameter.

- Expressions that use the select MIN, MAX, and COUNT functions:

```
SELECT MAX("#T".FIRSTNAME)
 FROM Database.CUSTOMER AS "#T"
 WHERE "#T".CUSTOMERID = custId
```

can be processed by ESQL functions that process the parent of the repeating field:

```
CREATE FUNCTION processMAX(IN custId)
BEGIN
 RETURN
 SELECT MAX("#T".FIRSTNAME)
  FROM Database.CUSTOMER AS "#T"
  WHERE "#T".CUSTOMERID = custId
END;
```

In the `.msgmap` file, call the ESQL function processMAX by using the element **custId** as a parameter.

- `.mfmap` files that use mfmap index variables in expressions:

```
e || "#I"
```

must be rewritten entirely in ESQL. By definition, there must be a complex repeating parent element, and this is not supported by ESQL functions.

- Expressions that use source expressions to compute values:

```
src_msg.e[src_msg.a]
```

must be rewritten by using `if` rows, msgmap:occurrence() functions, and ESQL functions:

```
for src_msg.e
 if
  condition msgmap:occurrence(src_msg/e) = src_msg/a
```

- For expressions that use index expressions to compute values:

```
src_msg.e["#I" +5]
src_msg.e[< 3]
```

the entire `.msgmap` file must be rewritten in ESQL, because the `.msgmap` files do not support indexed access to repeating fields.

- `.mfmap` files that use ROW expressions to compute values:

```
src_msg.e IN (1, 2, 3)
```

must be rewritten in ESQL, because `.msgmap` files do not support ESQL ROW expressions.

Restrictions on migrating maps that call ESQL

If there is a mismatch between the case that has been in the ESQL call in the message map, and the name of the routine defined in the ESQL file, an error is produced during migration of the message map. To prevent an error occurring during migration, ensure that the ESQL call in the message map uses the same case as the ESQL defined in the routines in the ESQL file. Alternatively, you can manually edit the message map after migration to call the ESQL routine with matching case.

Restrictions on migrating submaps

In Version 5.0 message maps, any complex element type can be the root of a submap. However, in Version 6.1, only a global element or a global attribute can be the root of a submap. When a Version 5.0 message map with a call to a submap with a non-global element as the map root is migrated, the submap is not migrated as a stand-alone submap. Instead, the call to the submap in the main message map is replaced by the migrated content of the submap. Alternatively, if the submap has a global element as the map root, the submap is migrated to a stand-alone Version 6.1 submap instead.

For Version 6.1, you must define reusable schema structures as global elements and types. If you have Version 5.0 submaps that use local elements, you must change the schema to add definitions of global elements for the local elements, and then use the new schema after migration. If the new global elements have the same name and type as the local elements, the Version 5.0 submaps do not have to be changed.

You must qualify a local element in a Version 5.0 submap with a namespace to ensure its successful migration to Version 6.1, because the global element that replaces it after migration must be qualified by the namespace. If your submap contains local elements, you must re-create the submap and re-create the call to the submap from the main message map.

The following table shows differences between the features that are supported in a submap for Version 5.0 and Version 6.1.

Version	Supported feature
Version 5.0	global elements and global attributes as map source
	global elements and global attributes as map target
	local elements and local attributes as map source
	local elements and local attributes as map target
Version 6.1	global elements, global attributes, and global types as map source
	global elements and global attributes as map target

Part 5. Appendixes

Appendix. Notices for WebSphere Message Broker

Read the legal notices for WebSphere Message Broker.

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web

sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information includes examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to

IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

(C) (*your company name*) (*year*). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. *_enter the year or years_*. All rights reserved.

Trademarks in the WebSphere Message Broker Information Center

Review the trademark information for WebSphere Message Broker.

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information at `www.ibm.com/legal/copytrade.shtml`.

Adobe is a registered trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Intel, Itanium, and Pentium are trademarks of Intel Corporation in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Index

Special characters

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Printed in USA